SOLVING COMPUTER VISION CHALLENGES WITH SYNTHETIC DATA

by

Weichao Qiu

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland

October 2020

Abstract

Computer vision researchers spent a lot of time creating large datasets. Yet there is still much information that is difficult to label. Detailed annotations like part segmentation and dense keypoint are expensive to annotate. 3D information requires extra hardware to capture. Besides the labeling cost, an image dataset also lacks the ability to allow an intelligent agent to interact with the world. As a human, we learn through interaction, rather than per-pixel labeled images. To fill in the gap of existing datasets, we propose to build virtual worlds using computer graphics and use generated synthetic data to solve these challenges.

In this dissertation, I demonstrate cases where computer vision challenges can be solved with synthetic data. The first part describes our engineering effort about building a simulation pipeline. The second and third part describes using synthetic data to train better models and diagnose trained models. The major challenge for using synthetic data is the domain gap between real and synthetic. In the model training part, I present two cases, which have differ-

ent characteristics in terms of domain gap. Two domain adaptation methods are proposed, respectively. Synthetic data saves enormous labeling effort by providing detailed ground truth. In the model diagnosis part, I present how to control nuisance factors to analyze model robustness. In the conclusion, I summarize future research directions which can benefit from synthetic data.

Thesis Readers

Dr. Alan L. Yuille (Advisor)
Bloomberg Distinguished Professor
Department of Cognitive Science & Computer Science
Johns Hopkins University

Dr. Gregory D. Hager
Mandell Bellmore Professor
Department of Computer Science
Johns Hopkins University

Dr. Yizhou Wang
Boya Professor
Department of Computer Science
Peking University

Acknowledgments

First, I want to thank my advisor, Prof. Alan Yuille, for his continuous support and guidance. I worked with him for many years starting at UCLA and benefited enormously from his attitude towards research. He focuses on his research goal, pursues critical challenges and not distracted by low hanging fruits. I am also grateful for his trust and support, when projects hit obstacles.

My projects were mostly done through teamwork. I want to thank all members of the DIVA team. Thank Prof. Greg Hager for leading this project. Thank Tae Soo Kim, Mike Peven, Zihao Xiao, Yi Zhang, and Jin Bai for working together through challenges. Thank Prof. Austin Reiter for initiating this project and set the direction. Thank our collaborators in other teams for trying out our data and gave feedback. Thank Kapil Katyal for helping us get motion capture data. Thank Prof. Misha Kazhdan for advices on shape annotation. Thank IARPA for funding this project.

Another team I extensively worked with is the unrealcv team. Thank core members: Fangwei Zhong, Siyuan Qiao. Thank Prof. Yizhou Wang for contin-

uous support. The project started when I was visiting Peking university in the 2015 summer and continued in the following years. Thank all users for giving feedback and thanks for the patience to all bugs and imperfection. Thank Le Lu, Xiaodi Hou, and many others for helpful discussions and advice.

I spent a happy internship in the Oculus Research lab (now Facebook Reality Lab), where I saw the very bright future of virtual reality. Thank my mentor Steven Lombardi for his guidance and Prof. Yaser Sheikh for giving me this opportunity.

I want to thank all my collaborators. I have the fortune to work with many excellent visiting students and master students of Alan's lab. I want to thank all of them: Qi Chen, Yi Zhang, Yiming Zuo, Qingfu Wan, Jiteng Mu, Pengfei Li, Xinyue Wei, Jialing Lyu, Jiahao Wang, Qihao Liu, Kejia Ren, Zizhang Li. They are all currently pursuing research in the computer vision field. Research is a Marathon. I hope they all enjoy research and have a very bright future in their career. I also want to thank collaborators: Lingxi Xie, Chenxi Liu, Qing Liu, Siyuan Qiao, Xiaohui Zeng, Michelle Shu, Prof. Wei Shen. I learned a lot through working with them. It is not possible to explore these exciting topics without their help.

I am fortune to meet many excellent mentors in my research journey. Without their help, I can not go as far. Thank Peng Wang, Xianjie Chen, Chunyu Wang, Prof. Hongjing Lv, Prof. Xinggang Wang, Prof. Jiayi Ma, Prof. Zhuowen

ACKNOWLEDGMENTS

Tu, Prof. Xiang Bai.

Thank all members of Alanlab for the support. Thank Lingxi, Jun, Adam, Yongyi, Yan, Chenxi, Zhuotun, Huiyu, Cihang, Yuyin, Zhishuai, Qing, Qi, Fengze, Yi, Yingda, Hongru, Chenxu, Jieru, Qihang, Yingwei, Yixiao, Zhuowan, Zihao, Chenglin, Yutong, Angtian, Chen.

I want to thank Prof. Yair Amir for the initial support of the toy robotic arm project. This project started as his course project and ended as a CVPR demo and paper. I want to thank my teammate Kaiyue Wu and Vicent Yang in the course for their contributions. Thank Prof. Ed Connor, Prof. Vishal Patel, Prof. Misha Kazhdan, Prof. Wei Shen, and Prof. Rene Vidal for serving my GBO committee.

Dedication

To my parents for their unconditional love.

Contents

CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

Data is the fuel for the recent success of computer vision (CV). Researchers spent great efforts to label images. Information such as bounding box and segmentation mask are easy to annotate. But there are more complex information to annotate, such as, semantic part segmentation, depth, etc. More importantly, meta information in an image is not labelled, like the lighting, material property, occlusion relationship. They are useful for many tasks. Computer graphics (CG) is a way to get these ground truth. It also offers the possibility of constructing *virtual worlds* in which: (i) an agent can perceive, navigate, and take actions guided by AI algorithms, (ii) properties of the worlds can be modified (e.g., material and reflectance), (iii) physical simulations can be performed, and (iv) algorithms can be learned and evaluated.

The rich ground truth and controllability of synthetic data can enable many

possibilities. For example, human parsing mainly focuses on 2D keypoint detection. With the help of synthetic data, researchers can study 3D shape estimation [4], semantic part segmentation [5], etc. Besides, the controllability of synthetic data enables studying model robustness and explaining a black-box model with data.

Synthetic data also enables research directions which are hard to perform using image and video datasets. Despite the superhuman performance for some tasks, models still lack some essential properties. As a human, we can 1) learn the visual world through interaction, in an unsupervised way, rather than given per-pixel annotation; 2) infer a visual concept through knowledge beyond vision; for example, a seat is defined by whether it is a place to sit; 3) combine concepts to create a new one, for example, combining human and horse to create the "centaur" concept. A virtual world gives us the tool to study these properties and can help build better models.

Despite these advantages, synthetic data suffers from the limitation of realism and diversity. Making models trained with synthetic data useful to real tasks requires domain adaptation. Domain gap not only exists between real and synthetic datasets, it also exists between different real datasets. So domain adaptation is not only important for synthetic data, but also important to make computer vision models generalizable. Domain adaptation can be achieved through generating better data or designing better models and train-

ing techniques. These two directions are both explored in this dissertation.

1.1 Why Using Synthetic Data

This section describes the difference between real and synthetic data and discusses the advantages and disadvantages of using synthetic data for computer vision tasks. Understanding this difference is the foundation to decide what computer vision research can and should involve synthetic data. This section focuses on computer graphics generated data; other generative models, such as GAN [6], share some similar properties.

The fundamental difference between real and synthetic data is that synthetic data are sampled from a virtual world. This gives us the power to sample from an infinite image space. On the contrary, image and video datasets, no matter how large, are sampled sparse data points. For example, given an image, if we want to get a new image from a slightly different viewpoint or lighting condition, this is not possible. But for an agent, no matter in a real or virtual world, this is achievable.

The advantages of synthetic data are summarized in the following section.

1.1.1 Advantages

The two major advantages of synthetic data are rich ground truth and controllability.

Rich ground truth. Synthetic data are generated from a virtual world, which we construct, so everything can be measured precisely. Synthetic data has the advantage to provide three types of ground truth: 1) more ground truth in existing datasets, 2) less-annotated meta information, 3) hard to annotate information.

First, synthetic data can provide more ground truth, which we can find in popular datasets. These ground truth can be the image class label, bounding box, keypoint, segmentation. More importantly, it can provide data that are laborious to annotate, like part segmentation, or requires special hardware, like depth. The goal is to reduce the labeling cost and to provide higher quality annotation. This type of data can be an addition to real data and be used together.

Second, it can provide meta-information, which is useful but not annotated due to the low application value. Besides commonly annotated information, there are much more "dark information" [7] of the scene. These information are helpful, but less annotated; for example, the lighting, material properties, occlusion relationship [8], etc.

This meta-information can be used in many innovative ways. It can be used

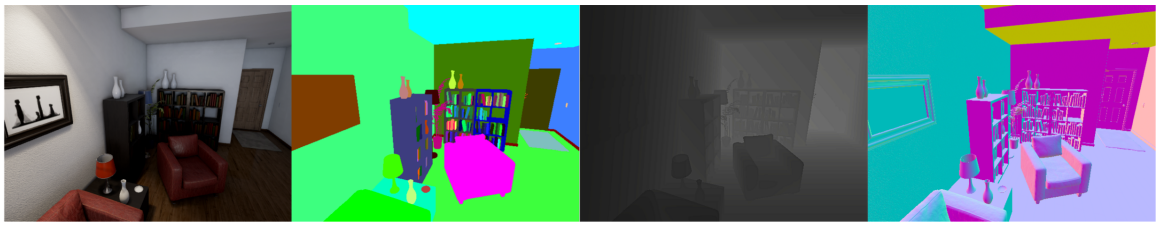

Figure 1.1: Ground truth from simulated data. From left to right are image, segmentation mask, depth and surface normal. Synthetic ground truth can capture tiny objects in the bookshelf, hard to annotate ground truth (surface normal), and other meta information of the scene, such as: lighting, material property, etc.

to analyze models, checking which information is helping a model to make a decision. It can be used for image captioning or VQA. The synthetic benchmark CLEVR dataset [9] is an example. It can also be used for a visual Turing test [10], where all sorts of questions can be asked.

Third, synthetic data can provide information that is beyond the image space, thus very hard to annotate. An example is future prediction through physics simulation. In [11], researchers built a vision model to predict whether a block tower is stable given its current configuration.

Commonly used ground truth from synthetic data can be seen from Fig. 1.1.

Controllability. The controllability enables two things: 1) understand the model through controllable data, 2) train the model through interaction.

Controllability can help us understand a model better. Popular benchmarks evaluate an algorithm on its average performance, which means each testing case contributes equally. If a model fails on a certain case, we would like to know whether it fails on similar cases. This requires sampling more cases de-

pending on existing failures. We also would like to know what is the boundary condition of a model. For example, if a model works well on regular lighting, whether it fails on low lighting and what is the threshold of lighting the model will fail.

Controllability also enables more learning paradigms, like curriculum learning or active learning. Active learning enables sampling more training data according to existing model performance. The vision module can also learn through completing a task, in which the supervision signal comes from the reward function of a task, rather than in the image level [12]. The low-cost of simulation and the controllability can well support these kinds of research.

Despite these advantages, there are many challenges that limit the wide adoption of synthetic data for solving real world challenges.

1.1.2 Limitations

A researcher, who is not familiar with how the computer graphics industry works, is usually first impressed by the photorealism of a rendered car or indoor scene. He then starts thinking whether synthetic data can be a cheap alternative for real data to save the burden of collecting images and annotating ground truth. This idea is attractive, but not practical for many scenarios. The limitations of the synthetic data are realism and diversity.

Realism. Currently, realism is easy to achieve for a small number of objects, especially when these objects are rigid[1]. Or it is also easy to cover a lot of objects with less realism [13]. But it is tough to achieve both. The realism depends on various factors, such as, whether the object is rigid or deformable, how complex the simulation is, what kind of lighting effects need to be considered, etc. Trying to mimic the real world entirely is the ultimate goal of CG, but quite expensive given the current technology.

Achieving full realism is not required for computer vision research; what level of realism depends on the task. If the goal is recognizing material property, then we need physics-based rendering, such as ray tracing, to generate subtle lighting effects (subsurface scattering, caustic effect, etc.). But in this scenario, we do not require the object shape to be semantically meaningful so that we can create random shapes [14] [15]. This means we can sacrifice the realism of the object shape.

On the other end of the spectrum, some tasks need to focus more on behavior realism, rather than low-level visual details. In this case, visual realism are less important, and the semantic meaning is more important. Real-time video games [16] are more suitable for this case, although the rendering quality is worse than offline rendering. Because the game company spent a lot of resources to reconstruct virtual city scenes, model car behaviors and motion

[1]Realistic 3D models can be purchased, for example, from `https://turbosquid.com`

capture human activities. So even the rendering quality is from a low-quality rasterization engine, it is the most suitable resource.

Diversity. The diversity of a virtual world includes 3D model diversity and behavior diversity. The diversity is essential to prevent overfitting and important for domain adaptation.

The engineering cost is the main limiting factor. Solving this requires a tight collaboration with the game and movie industries. It can be reusing 3D assets from a CG movie or a video game. It can also be utilizing photogrammetry tools from the CG industry to create realistic 3D models. The diversity issue makes it difficult, if not impossible, to make a virtual replicate of ImageNet [17].

Diversity is much easier to achieve in the real world for some tasks, like collecting images from the internet [17] or bringing robots to Airbnb rooms for different scenes [18]. It is easy to get a very realistic car in CG, but if the goal is to get 100 different instances of a sedan, taking photos in a parking lot is far more manageable.

The synthetic data can also be very diverse through procedural generation. This means generating random shapes or random walking patterns through algorithms. This technique has been used for generating random shapes for cognitive science [14] or for studying low-level vision algorithms, like stereo [15]. In these scenarios, although generated data has less semantic meaning, the

	Image / Video Dataset	Human / Robot	Synthetic Data
Ground Truth	Expensive or Requires Hardware	No	Free and Rich
Controllability	No	Slow, Limited Control	Fast, Full Control
Data Diversity	Diverse	Diverse	Limited or Procedural

Table 1.1: Comparison between real dataset, a real agent and a virtual agent. Synthetic data provides a virtual world which a virtual agent can interact with.

data is still diverse and useful.

In Tab. 1.1, we summarize the difference between a real dataset, a real agent, and a virtual agent. In summary, synthetic data should not be considered as a replacement for real data. These unique properties of the synthetic data need to be utilized to justify the high engineering efforts. Synthetic and real data can be complementary to each other.

1.2 Outline

The dissertation consists three parts: 1) Build simulation infrastructure to utilize synthetic data, 2) Use synthetic data to train better models, 3) Use synthetic data to diagnose and understand a model.

Part I is simulation infrastructure. Chapter 2 [19] describes our engineering efforts for building a synthetic data generation pipeline.

Part II is model training. In this part, I demonstrate how to use synthetic data to solve challenges which are difficult for real data.

In Chapter 3 [20], we use synthetic data to train a keypoint detector and

pose estimator for a toy robotic arm. This $40 robotic arm does not have any joint angle sensors, which means it can not be programmably controlled. Our vision module can estimate joint angles from a single image and control this arm to accomplish tasks. No training data for this toy robotic arm exists and annotating 3D configuration from images is tedious. We use synthetic data to solve this challenge. Without labeling any images, all training data came from rendering the articulated CAD model. Because the geometry of the CAD model and the real arm are exactly the same, this enables use to design a domain adaptation method based on geometry constraint.

In Chapter 4 [21], we use synthetic animal images to train keypoint detectors for animals. Different from human keypoint, which the industry spends a lot of resources to annotate, animal keypoint annotations are harder to find. So we solve this lack of data challenge with generated synthetic ground truth. We have animal CAD models, but the appearance is very different from the real world animal, in terms of details and diversity. In order to achieve domain adaptation, we design a consistency-based semi-supervised learning method and use it together with domain randomization. This method enables us to train animal keypoint detectors without real image annotation.

Domain difference exists not only between synthetic and real data but also between different real-world datasets. We can consider the domain adaptation challenge as a domain generalization challenge. Some preliminary re-

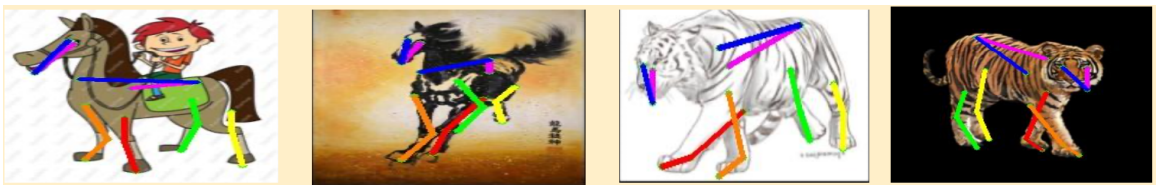

Figure 1.2: Synthetic data trained model can generalize well to other domains. While we are solving the domain adaptation issue, we are improving domain generalization as well. This figure shows model trained using synthetic animal images can generalize well to cartoon animal images.

sult shows when aiming for domain adaptation, the generalization between domains also improves. Fig. 1.2 shows a model trained using synthetic data can better generalize to different domains than using real data.

Part III is model diagnosis. Despite the great success of computer vision models, we still lack a deep understanding of these models. There are two questions we are particularly interested in; first, whether the model is robust to corner cases or physically feasible adversarial attack; second, whether the model is using the correct cues for inference rather than overfitting. We can tweak the rendering parameters of an image to check whether the model is sensitive to these parameters. For example, we can tweak the background of the input to an action classification network. In this way, we can understand whether the model uses background information rather than the motion pattern to classify the action.

In Chapter 5 [22], we use synthetic data to analyze whether a stereo algorithm is robust to hazardous conditions, e.g. transparency, reflection, lack of textures. This is done through changing material properties of synthetic

data. In Chapter 6 [23], we study whether activity classification methods use the human motion pattern or overfit to context objects and background. Meta information from the virtual world enables us to study the invariant and equivariant properties of a model. These information are difficult to tweak for a real image and annotating them is prohibitively expensive.

In Chapter 7, I summarize this dissertation and describe some potential directions that can benefit from synthetic data.

Part I

Infrastructure

Chapter 2

Data Generation Infrastructure

This chapter summaries our simulation infrastructure for generating synthetic images and ground truth.

2.1 Introduction

Realistic virtual worlds are created by the movie and game industry to tell stories that are difficult to see in the real world. These photo-realistic virtual worlds are also useful for research, they can be used for many tasks that the real world is not capable of, such as generating large number of images with hard-to-get annotations [24], simulate images of extreme weather conditions [25], stress test a vision algorithm by creating hazardous factors [22] and do expensive robotics training [26].

Constructing a *virtual world* from a high-fidelity 3D video game or CG movie is attractive for computer vision researchers. This motives the creation of OpenAI universe, Malmo, VizDoom, etc. But modifying a video game or CG movie has two limitations. First, open source video games, such as Doom, have limited visual realism, while the newest video games such as GTA do not have good APIs to access its internal data and sometimes have license issues. Second, the modifications of one video game can not be transfered to others and needs to be redone case by case. In order to use a video game as a virtual world for computer vision researchers, some extensions need to be done: (1) the video game needs to be programmly accessible through an API, so that an AI agent can communicate with it (2) the information in the virtual worlds need to be extracted to achieve certain tasks, such as ground truth generation or giving rewards based on the action of agents.

Constructing a virtual world using a game engine can solve the problems in modifying a game. *Game engine* is a software framework for creation of video games. If the extension is done in the game engine, then the game engine can be used to produce video game (virtual world) that can meet the requirements of researchers. Unreal Engine (UE4) is a popular game engine for creating high-fidelity video game and one of the best choices for virtual reality developments. The rich 3D resources and full access to its source code motivates us to extend it for creating virtual worlds that are useful for researchers. This idea

leads to the creation of UnrealCV.

UnrealCV extends UE4 to make it able to create virtual worlds that can meet requirements of researchers. Two additional features are provided. First, it provides a communication layer, so that a program can communicate with UE4 to extract information. Second, commonly used computer vision features are provided, such as ground truth generation shown in Fig. 1.1. A preliminary version of UnrealCV is published in [27]. The design and implementation of UnrealCV satisfies these requirements. (1) Easy to install and use and can support major platforms and languages for research, such as Python and Linux. Researchers are not game developers and UnrealCV can be used without knowing anything about UE4 and game design. (2) Extensible to include more features that researchers need. (3) Compatible with the up-to-date version of Unreal Engine to benefit from the improvement from the upstream.

Virtual world has been attractive for AI researchers since the birth of AI, but the poor visual realism limited the adoption. This has been changed in the past few years. The visual realism and cost of virtual world is approaching a critical point that can be extremely useful for computer vision research. Getting a high quality computer graphics is difficult and expensive before. This cost has been hugely reduced by 3D marketplace, better 3D reconstruction tools. The prosper of virtual reality makes more people create and share high quality 3D contents and better tools to support that. Unreal Engine is one of

the best choice for virtual reality development and very open to the community. It hugely reduce the cost for accessing high-quality computer graphics. UnrealCV provides a bridge between computer vision and Unreal Engine to help researchers easily utilize these high quality computer graphics resource.

2.2 Highlights of UnrealCV

UnrealCV provides a set of tools to make the creation and sharing of virtual worlds easier. It extends Unreal Engine to make the popular game engine able to construct virtual worlds that can meet requirements of researchers out-of-the-box.

Easy to use Game binaries created by UnrealCV are provided, so that researchers can start using a virtual world without any knowledge of UE4. Using a virtual world is as simple as download a game and run it. These game binaries are organized with a model zoo.

UnrealCV supports popular platforms (Linux, Windows) and Python. Any language that can use socket can be easily supported. It is compatible with a wide range of UE4 versions. It can also be used together with UE4 Editor to design new virtual worlds.

In order to make the virtual environment easy to deploy, we also provide docker images to simplify the installation and creation of distributed learning

systems. The docker image is also used to do automatically building, testing of the software.

Powerful and extensible UnrealCV makes the information of a virtual world accessible through URI (unique resource identifier), each resource in the virtual world is associated with a URI. For example, the object location can be accessed with `/object/[id]/location`. More resources of the virtual world can be exposed by extending the UnrealCV.

The URI is defined in a hierarchical modular way. For example, `/light/[name]/intensity` is used for the light intensity, a new URI `/light/[name]/color` to access the light color can be added without affecting existing ones.

Some information of the virtual world is not directly accessible, but needs to be computed by UnrealCV, for example `/camera/[id]/object_mask` is the object segmentation mask shown in Fig. 1.1. There are also some abstract resource for special purposes, such as `/action/keyboard` to simulate a keyboard input. The URI is used in the RESTful commands of UnrealCV which is described in Sec. 2.3.2. The commands can be used to achieve simple tasks, such as generating an image dataset, or be combined to achieve a complex task such as reinforcement learning, which is described in Sec. 2.4.

We are not only actively working with our collaborators to add new features,

but also provide documentation to show how to implement new commands[1] for the system. UnrealCV is open-source and can be extended by anyone. Unit tests are provided to make this easier.

Well documented and tested Tutorials are provided from simple task such as image dataset generation to complex task such as reinforcement learning. API documentation of UnrealCV is provided to help others extend the software.

A lot of tutorials of designing video games can be found for Unreal Engine, but not very relevant for the research purpose. In the UnrealCV project, we also organized relevant tutorials to research. A list of research projects that using virtual worlds for computer vision is maintained by the team as a separate project[2] to help researchers to compare and find useful tools.

Model Zoo Inspired by the idea of model zoo from Caffe, we created a model zoo for virtual worlds. It is a place for sharing virtual worlds.

The model zoo make it easy to use virtual world without the knowledge of UE4 and no need to purchase 3D models. According to the license, it is not possible to distribute source content directly. But the binary can be released, the link to the original source content will be provided, so that researchers require the source content can buy the 3D models and use them within the UE4 editor.

[1] http://docs.unrealcv.org/en/develop/plugin/develop.html
[2] https://github.com/unrealcv/synthetic-computer-vision

Initially, six virtual worlds are released for data generation and algorithm diagnosis [27] [22]. The instruction for releasing new virtual environments are included to encourage the community to creating new virtual worlds and sharing them[3].

A virtual world can be used for many different tasks. Sharing virtual worlds is a new concept for computer vision dataset. We will also share images we generated from the virtual world. Moreover, scripts are provided to generate more images from the virtual scene. The virtual world also enable new tasks that traditional image/video dataset can not support, such as reinforcement learning and active vision. UnrealCV is an underlying tool to make virtual worlds easy to construct and share.

2.2.1 Compare to Related Software

Unreal Engine has been used in a few research projects [28] [26]. The game engine was extended to meet the requirement of a specific research project. But customization of Unreal Engine makes it harder to adapt to newer version. There is no model zoo for sharing virtual worlds. so the usage of UE4 requires a considerate amount of preparation.

We aim to provide a flexible and extensible API that can be combined to achieve different goals. AirSim [28] and CARLA [29] focus on providing a com-

[3]http://docs.unrealcv.org/en/develop/plugin/package.html

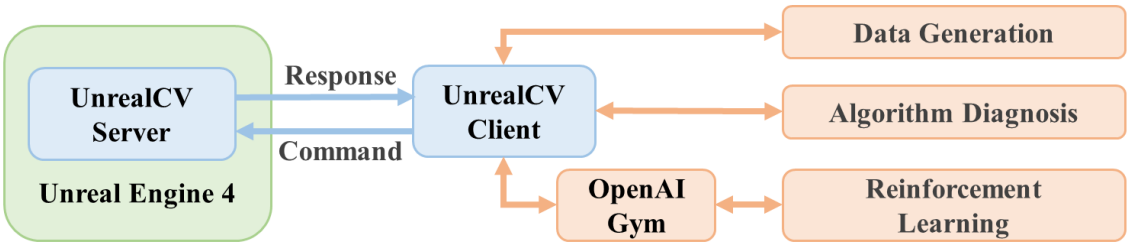

Figure 2.1: UnrealCV consists of the UnrealCV server which is embedded into a game during compilation. External programs use the UnrealCV client to communicate with UE4 games. See text for details.

plete simulation environment built with UE4 for a task. Our tool aims to provide modules that can be integrated with any virtual worlds built with the game engine.

2.3 Architecture

UnrealCV provides an easy way to read and modify resources of a virtual world. The user imports the *UnrealCV client* as a library into his code. The UnrealCV client will be used to request information from the virtual world, for example, reading the 3D location of an object. This is done through sending an *UnrealCV command* to the virtual world. The *UnrealCV server* processes the command and returns requested information. This procedure is shown in Fig. 2.1.

2.3.1 UnrealCV Server and Client

UnrealCV has two components, the server and the client. The server uses the C++ API of Unreal Engine to access information of the virtual world. The client is a library to communicate with the server by sending commands and parsing responses. The server and client communicates using TCP. The communication protocol is defined by a set of UnrealCV commands, described in Sec. 2.3.2.

The server implements features useful for computer vision researchers, such as ground truth generation. These information can be computed from the internal data of Unreal Engine, but they are missing from UE4.

The client is a small library that can be easily integrated into other code. We demonstrate this with a simple image generation tutorial which uses the client in a short script and a complex reinforcement learning demo, which involves tensorflow and needs to be run for days in Sec. 2.4. Python version of UnrealCV client is provided and can be installed through the package manager. Experimental support for MATLAB is provided. We document the protocol used in the client implementation, making it easier to implement a client for a different languages that can support socket.

The communication between the server and the client are two ways. The client requests information from the server and the server can also send a message to the client to notify an event, making it easier to design tasks which

requires triggering events, for example, sending collision notification in reinforcement learning. Currently only one concurrent client is supported. Multiple clients will be useful to simulate multiple intelligent agents interacting in one virtual environment. This will be our future work.

2.3.2 Command System

The resources of a virtual world are exposed through a set of commands. The command is designed in a RESTful style. A typical command, for example, `vset /camera/[id]/location [x] [y] [z]`, consists of three parts. The first part is the operation to perform, it can be either `vget` or `vset`. The second part is an URI indicating the resource of the 3D world to operate with. The third part is extra parameters, such as the new location of an object.

These commands are documented[4] and tutorials are provided to show how to combine commands to achieve certain research tasks.

The command is an abstraction of what is needed from a virtual world. This concept can be applied to other softwares, creating a unified API for the research community, which is beyond Unreal Engine.

[4]http://docs.unrealcv.org/en/develop/reference/commands.html

2.3.3 Virtual World Creation using UnrealCV

As a game engine, Unreal Engine provides tools to package a game. *Packaging* produces a video game binary containing rendering and physics simulation code, 3D models and tasks for players. UnrealCV adds computer vision related code to the game binary during packaging, making the binary can be used as a virtual world. The information of the virtual world will be exposed through UnrealCV commands.

2.3.4 UE4 Editor Plugin

Researchers will not be satisfied with existing virtual worlds once they have an idea that existing ones cannot support. For example, we may want to place a glass door into the room to see whether the robot can successfully avoid it, but no existing virtual worlds provide such a setup. Therefore it is necessary to have a tool for designing virtual worlds.

Unreal Engine Editor (UE4Editor) is a useful tool for editing a 3D environment, similar to 3DS max and Maya. It can be used to add/delete objects, modify details of the scene. UnrealCV does not provide the complex 3D editing that UE4Editor can provide, but UnrealCV can be used as a plugin of the editor, enabling researchers to combine the power of both. With UnrealCV plugin installed, UnrealCV server code will run inside the UE4Editor to extend its

functions.

2.4 Applications and Examples

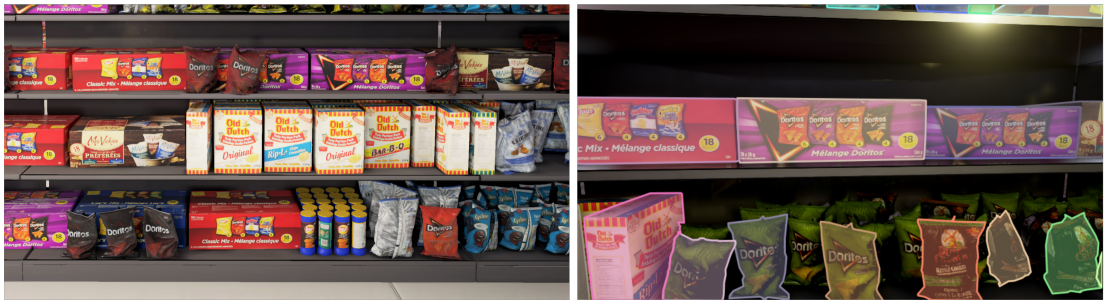

Figure 2.2: A virtual supermarket shelf and its annotation visualized by MSCOCO API. The left is the synthetic image and the right is the segmentation mask, see [1] for more details.

Data Generation A virtual world can be used to generate large amount of images with ground truth. A virtual world can also produce ground truths that are hard to annotate in a real image. For example, for the grocery shelf image in Fig. 2.2, it is very time consuming to annotate the object mask of individual items. It is even more challenging to annotate the occluded regions of objects. In order to get a benchmark image dataset of supermarket for training, a virtual supermarket was constructed in [1] using UnrealCV. The object placement, lighting were randomized to increase the variety of the data and prevent over-fitting for training object detectors. A tool for randomly placing objects on the self is provided, which can be modified to randomly generate other scenes.

This tool is also included in UnrealCV. The object detector is trained only with synthetic data and can work well on real images, see [1] for detailed results. The synthetic image and visualization of ground truth is shown in Fig. 2.2.

Algorithm Diagnosis The preliminary paper of UnrealCV [27] gave an example of diagnosing an object detector. It showed the performance degeneration when viewpoint varied.

A more extended diagnosis study was done for stereo algorithms in [22]. In virtual world, it is convenient to adjust material properties, such as texture, surface reflectance and transparency to create certain degrees of hazardous regions to stereo matching. This enables us to densely sample the degrees of hazardous factors to analyze their effect on stereo algorithms. Findings on synthetic datasets are further verified by constructing small real world datasets tuned to these precise degrees of hazard.

Reinforcement Learning

It is expensive, time-consuming and in some cases dangerous to train agents by trial-and-error in real world. So it is necessary to train agents in a virtual environment. But most of available virtual environments are far different from the real world in terms of appearance and physics. Realistic virtual environment is important to help robots evolve from playing games to solving real-world problems, like grabbing object or visual navigation.

UnrealCV provides a new way to construct realistic virtual environments.

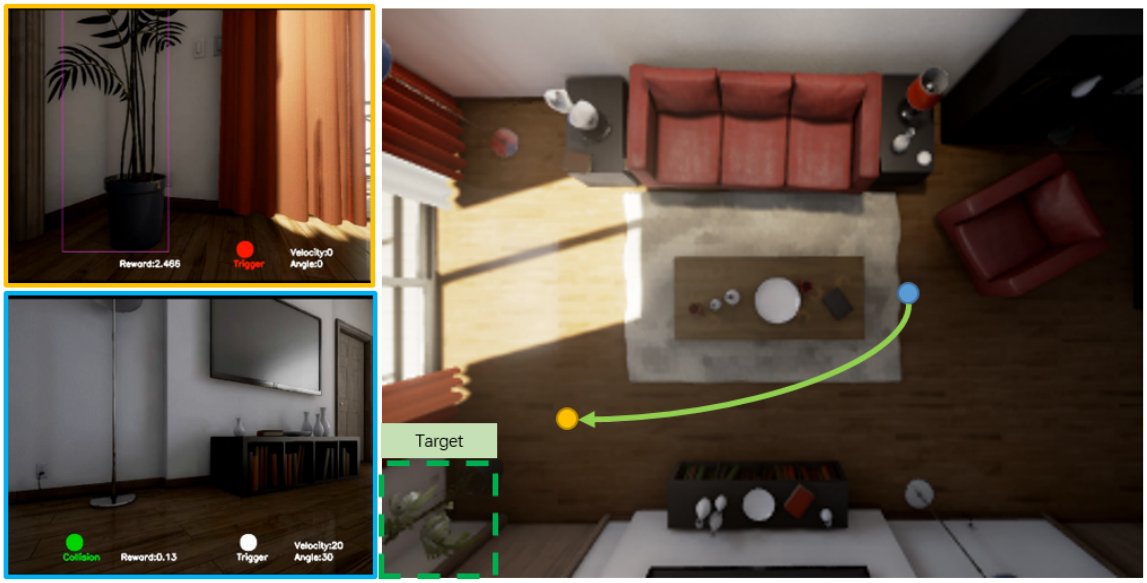

Figure 2.3: Visual navigation in a realistic room. The top view shows the room layout, target object and a path learned by DRL to find target while avoiding collision. The navigation starts at a target-unseen place. The left images show agent's first-person view at the begin and end of the path.

The high-fidelity rendering and advanced physical engine make it easier to transfer from virtual to real world. We provide an UnrealCV based OpenAI Gym interface to help researchers integrate RL algorithm with the virtual world easily. This reinforcement learning interface can be used without knowledge of UE4. Rich ground truths from UnrealCV make it flexible to design reward function for new tasks. To help users get started with the installation and usage, we provide a tutorial and a visual navigation example shown in Fig. 2.3, in our project page.

2.5 Conclusion

This chapter has presented a tool called UnrealCV which can be plugged into the game engine UE4 to help construct realistic virtual worlds from the resources of the game, virtual reality, and architecture visualization industries. These virtual worlds allow us to access and modify the internal data structures enabling us to extract groundtruth, control an agent, and train and test algorithms. Using virtual worlds for computer vision still has challenges, e.g., the variability of 3D content is limited, internal structure of 3D mesh is missing, realistic physics simulation is hard, and transfer from synthetic images remains an issue. But more realistic 3D contents will be available soon due to the advance of technology and the rising field of VR. As an industry leader, UE4 will benefit from this trend. UnrealCV is an open-source tool and we hope other researchers will use it and contribute to it.

Part II

Model Training

Chapter 3

Toy Robotic Arm Control using Synthetic Training Data

In this chapter, we build a simulator to generate ground truth for a toy robotic arm, then train a vision algorithm to parse 3D configuration of the arm. This toy arm does not have any joint angle sensor, but with the help of computer vision, we can control it to accomplish simple tasks. In order to solve the domain gap between real and synthetic, we design a domain adaptation algorithm using geometry constraint. We save enormous human labor by generating ground truth from the CAD model, rather than annotating dense keypoints and joint angles by hand.

3.1 Introduction

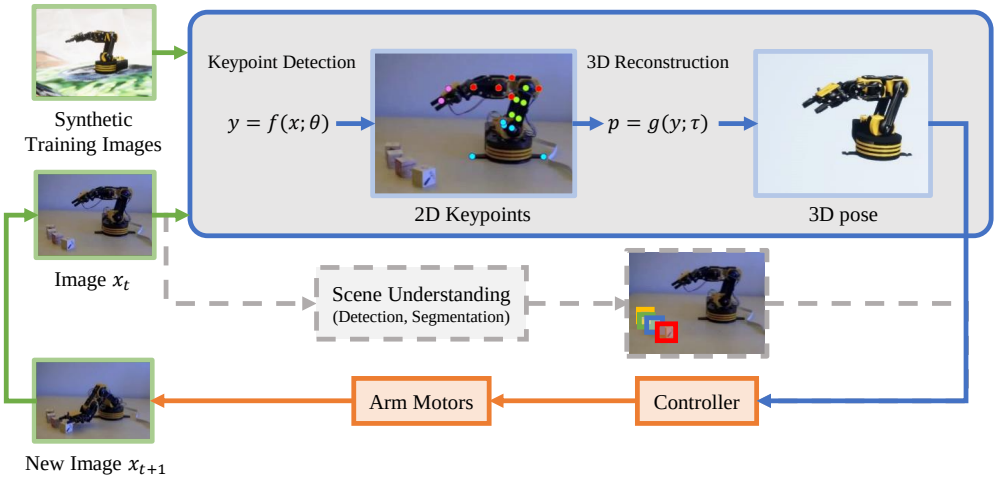

Figure 3.1: An overview of our system (best viewed in color). The goal is to accomplish tasks using camera as the only sensor. The vision module detects 2D keypoints of the arm and then computes its 3D pose. The control module uses the 3D pose estimation to determine the next move and feeds it to the motors. In our setup, scene understanding is achieved by directly providing the 3D location of targets.

Precise and agile robotic arms have been widely used in the assembly industry for decades, but the adaptation of robots to domestic use is still a challenging topic. This task can be made much easier if vision input are provided and well utilized by the robots. A typical example lies in autonomous driving [30]. In the area of robotics, researchers have paid more and more attentions to vision-based robots and collected large-scale datasets, *e.g.*, for object grasping [31] [32] and block stacking [33]. However, the high cost of configuring a robotic system largely limits researchers from accessing these interesting topics.

This work aims at equipping a robotic system with computer vision algorithms, *e.g.*, predicting its real-time status using an external camera, so that researchers can control them with a high flexibility, *e.g.*, mimicking the behavior of human operators. In particular, we build our platform upon a low-cost robotic arm named OWI-535 which can be purchased from Amazon[1] for less than $\$40$. The downside is that this arm has no sensors and thus it totally relies on vision inputs[2] – on the other hand, we can expect vision inputs to provide complementary information in sensor-equipped robotic systems. We chose this arm for two reasons. (i) **Accessibility**: the cheap price reduces experimental budgets and makes our results easy to be reproduced by lab researchers (poor vision people :( ). (ii) **Popularity**: users around the world uploaded videos to YouTube recording how this arm was manually controlled to complete various tasks, *e.g.*, picking up tools, stacking up dices, *etc*. These videos were captured under substantial environmental changes including viewpoint, lighting condition, occlusion and blur. This raises real-world challenges which are very different from research done in a lab environment.

Hence, the major technical challenge is to **train a vision algorithm to estimate the 3D pose of the robotic arm**. Mathematically, given an input image x, a vision model $\mathbb{M} : \mathrm{p} = \mathrm{h}(\mathrm{x}; \boldsymbol{\theta})$ is used to predict p, the real-time

[1] https://www.amazon.com/dp/B0017OFRCY/

[2] Even when the initialized status of the arm is provided and each action is recorded, we cannot accurately compute its real-time status because each order is executed with large variation – even the battery level can affect.

3D pose of the arm, where $\boldsymbol{\theta}$ denotes the learnable parameters, *e.g.*, in the context of deep learning [34], network weights. Training such a vision model often requires a considerable amount of labeled data. One option is to collect a large number of images under different environments and annotate them using crowd-sourcing, but we note a significant limitation as these efforts, which take hundreds of hours, are often not transferable from one robotic system to another. In this chapter, we suggest an alternative solution which borrows a 3D model and synthesizes an arbitrary amount of labeled data in the virtual world with almost no cost, and later adapts the vision model trained on these virtual data to real-world scenarios.

This falls into the research area of domain adaptation [35]. Specifically, the goal is to train $\mathbb{M}$ on a virtual distribution $\mathrm{x}^{\mathrm{V}} \sim \mathcal{P}^{\mathrm{V}}$ and then generalize it to the real distribution $\mathrm{x}^{\mathrm{R}} \sim \mathcal{P}^{\mathrm{R}}$. We achieve this goal by making full use of a strong property, that the spatial relationship between keypoints, *e.g.*, the length of each bone, is fixed and known. This is to say, although the target distribution $\mathcal{P}^{\mathrm{R}}$ is different from $\mathcal{P}^{\mathrm{V}}$ and data in $\mathcal{P}^{\mathrm{R}}$ remain unlabeled, the predicted keypoints should strictly obey some geometric constraints $\boldsymbol{\tau}$. To formulate this, we decompose $\mathbb{M}$ into two components, namely $\mathbb{M}_1 : \mathrm{y} = \mathrm{f}(\mathrm{x}; \boldsymbol{\theta})$ for keypoint detection and $\mathbb{M}_2 : \mathrm{p} = \mathrm{g}(\mathrm{y}; \boldsymbol{\tau})$ for 3D pose estimation, respectively. Here, $\mathbb{M}_2$ is parameter-free and thus cannot be optimized, so we train $\mathbb{M}_1$ on $\mathcal{P}^{\mathrm{V}}$ and hope to adapt it to $\mathcal{P}^{\mathrm{R}}$, and y becomes a hidden variable. We apply an iterative

algorithm to infer $\mathbf{p}^\star = \arg\max_{\mathbf{p}} \int \Pr(\mathbf{y}; \boldsymbol{\theta} \mid \mathbf{x}) \cdot \Pr(\mathbf{p}; \boldsymbol{\tau} \mid \mathbf{y}) \, \mathrm{d}\mathbf{y}$, and the optimal $\mathbf{y}^\star$ determined by $\mathbf{p}^\star$ serves as the guessed label, which is used to fine-tune $\mathbb{M}_1$. Eventually, prediction is achieved without any annotations in the target domain.

We design two benchmarks to evaluate our system. The first one measures pose estimation accuracy, for which we manually annotate two image datasets captured in our lab and crawled from YouTube, respectively. Our algorithm, trained on labeled virtual data and fine-tuned with unlabeled lab data, achieves a mean angular error of 4.81°, averaged over 4 joints. This lays the foundation of the second benchmark in which we create an environment for the arm to accomplish a real-world task, *e.g.*, touching a specified point. Both quantitative (in distance error and success rates) and qualitative (demos are provided in the supplementary material) results are reported. Equipped with reinforcement learning, our vision-based algorithm achieves comparable accuracy with human operators. All our data and code have been released at our website, `https://craves.ai`.

In summary, the contribution of this chapter is three-fold. First, we design a complete framework to achieve satisfying accuracy in task accomplishment with a low-cost, sensor-free robotic arm. Second, we propose a vision algorithm involving training in virtual environment and domain adaptation, and verify its effectiveness in a typical multi-rigid-body system. Third, we develop a plat-

form with two real-world datasets and a virtual environment so as to facilitate future research in this field.

3.2 Related Work

Vision-based Robotic Control

Vision-based robotic control is attracting more and more attentions. Compared with conventional system relying on specific sensors, *e.g.*IMU and rotary encoder, vision has the flexibility to adapt to complex and novel tasks. Recent progress of computer vision makes vision-based robotic control more feasible. Besides using vision algorithms as a perception module, researchers are also exploring training an end-to-end control system purely from vision [36] [12] [37]. To this end, researchers collected large datasets for various tasks, including grasping [31] [32], block stacking [33], autonomous driving [30] [38], *etc*.

On the other hand, training a system for real-world control tasks is always time-consuming, and high-accuracy sensor-based robots are expensive , both of which have prevented a lot of vision researchers from entering this research field. For the first issue, people turned to use simulators such as MuJoCo [39] and Gazebo [40] so as to accelerate training processes, *e.g.*, with reinforcement learning, and applied to real robots, *e.g.*, PR2 [41], Jaco [42] and KUKA

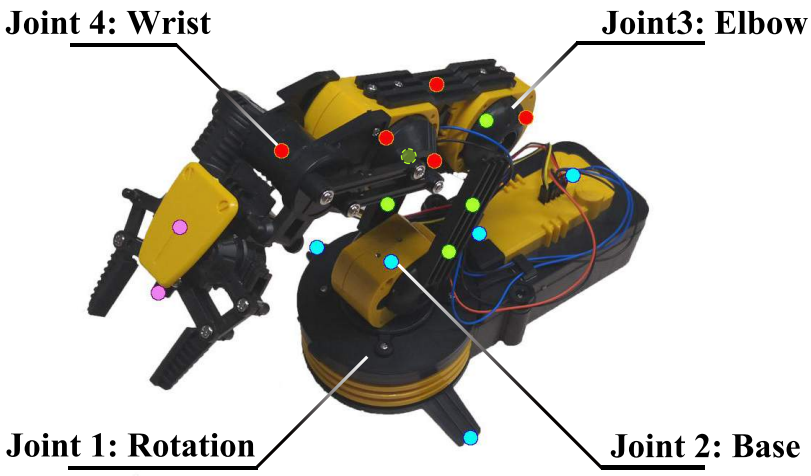

Figure 3.2: Here shows the 4 joints and 17 keypoints of OWI-535 used in our experiment. Each joint is assigned a specific name. Color of keypoint correspond to the part to which it belongs.

IIWA [43](ranging from $\$50,000$ to $\$200,000$). For the second issue, although low-cost objects (*e.g.*, toy cars [44]) have been used to simulate real-world scenarios, low-cost robotic arms were rarely used, mainly due to the limitation caused by the imprecise motors and/or sensors, so that conventional control algorithms are difficult to be applied. For instance, Lynxmotion Arm is an inexpensive ($\$300$) robotic arm used for training reinforcement learning algorithms [45] [46]. The control of this arm was done using a hybrid of camera and servo-motor, which provides joint angle. This chapter uses an even cheaper ($\$40$) and more popular robotic arm named OWI-535, which merely relies on vision inputs from an external camera. To the best of our knowledge, this arm has never been used for automatic task accomplishment, because lacking of sensors.

Computer Vision with Synthetic Data

Synthetic data have been widely applied to many computer vision problems in which annotations are difficult to obtain, such as optical flow [24], object tracking [47] [48] [49], human parsing [5], VQA [9], 6-D pose estimation [50] [51], semantic segmentation [52], *etc*.

Domain adaptation is an important stage to transfer models trained on synthetic data to real scenarios. There are three major approaches, namely, domain randomization [53] [54], adversarial training [55] [56] [57] [58] and joint supervision [59]. A more comprehensive survey on domain adaptation is available in [35]. As an alternative solution, researchers introduced intermediate representation (*e.g.*, semantic segmentation) to bridge the domain gap [60]. In this work, we focus on semi-supervised learning with the assistance of domain randomization. The former method is mainly based on 3D priors obtained from modeling the geometry of the target object [61] [62]. Previously, researchers applied parameterized 3D models to refine the parsing results of humans [63] [64] or animals [65], or fine-tune the model itself [62]. The geometry of a robotic system often has a lower degree of freedom, which enables strong shape constraints to be used for both purposes, *i.e.*, prediction refinement and model fine-tuning.

3.3 Approach

3.3.1 System Overview

We aim at designing a vision-based system to control a sensor-free robotic arm to accomplish real-world tasks. Our system, as illustrated in Figure 3.1, consists of three major components working in a loop. The first component, *data acquisition*, samples synthetic images x from a virtual environment for training and real images from an external camera for real-time control. The second component is *pose estimation*, an algorithm $\mathrm{p} = \mathrm{h}(\mathrm{x}; \boldsymbol{\theta})$ which produces the 3D pose (joint angles) of the robotic arm (see Figure 3.2 for the definition of four joints). The third component is a *controller*, which takes p as input, determines an action for the robotic arm to take, and therefore triggers a new loop.

Note that data acquisition (Section 3.3.2) may happen in both virtual and real environments – our idea is to collect cheap training data in the virtual domain, train a vision model and tune it into one that works well in real world. The core of this chapter is pose estimation (Section 3.3.3), which is itself an important problem in computer vision, and we investigate it from the perspective of domain transfer. While studying motion control (Section 3.3.4) is also interesting yet challenging, it goes out of the scope of this chapter, so we setup a relatively simple environment and apply an reinforcement learning algorithm.

3.3.2 Data Acquisition

The OWI-535 robotic arm has 5 motors named *rotation*, *base*, *elbow*, *wrist* and *gripper*. Among them, the status of the *gripper* is not necessary for motion planing and thus it is simply ignored in this chapter. The range of motion for the first 4 motors are $270°$, $180°$, $300°$ and $120°$, respectively.

In order to collect training data with low costs, we turn to the virtual world. We download a CAD model of the arm with exactly the same appearance as the real one which was constructed using Unreal Engine 4 (UE4)[3]. Using Maya, we implement its motion system which was not equipped in the original model. The angle limitation as well as the collision boundary of each joint is also manually configured. The CAD model of OWI-535 has $170{,}648$ vertices in total, among which, we manually annotate 17 visually distinguishable vertices as our keypoints, as shown in Figure 3.2. This number is larger than the degree-of-freedom of the system (6 camera parameters and 4 joint angles), meanwhile reasonably small so that all keypoints can be annotated on real images for evaluation. The images and annotations are collected from UE4 via UnrealCV [19].

We create real-world dataset from two sources for benchmark and replication purpose. The first part of data are collected from an arm in our own lab, and we maximally guarantee the variability in its pose, viewpoint and background. The second part of data are crawled from YouTube, on which many

[3]https://3dwarehouse.sketchup.com/model/u21290a3e-f8ef-46da-985c-9aa56b0dee53/Maplin-OWI-ROBOTIC-ARM

users uploaded videos of playing with this arm. Both subsets raise new challenges which are not covered by virtual data, such as motion blur and occlusion, to our vision algorithm, with the second subset being more challenging as the camera intrinsic parameters are unknown and the arm may be modded for various purposes. We manually annotate the 17 keypoints on these images, which typically takes one minute for a single frame.

More details of these datasets are covered in Section 3.4.

3.3.3 Transferable 3D Pose Estimation

We first define some terminologies used in this chapter. Let $\mathbf{p}$ define all parameters that determine the arm's position in the image. In our implementation, $\mathbf{p}$ has 10 dimensions: 6 camera extrinsic parameters (location, rotation) and 4 angles of motors. $\mathbf{y} \in \mathbb{R}^{17\times 2}$ and $\mathbf{z} \in \mathbb{R}^{17\times 3}$ are the locations of keypoints in 2D and 3D, respectively. Both $\mathbf{y}$ and $\mathbf{z}$ are *deterministic* functions of $\mathbf{p}$.

The goal is to design a function $\mathbf{p} = \mathbf{h}(\mathbf{x}; \boldsymbol{\theta})$ which receives an image $\mathbf{x}$ and outputs the pose vector $\mathbf{p}$ which defines the pose of the object. $\boldsymbol{\theta}$ denotes the learnable parameters, *e.g.*, network weights in the context of deep learning. The keypoints follow the same geometry constraints in both virtual and real domains. In order to fully utilize these constraints, we decompose $\mathbf{h}(\cdot)$ into two components, namely, 2D keypoint detection $\mathbb{M}_1 : \mathbf{y} = \mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ and 3D pose estimation $\mathbb{M}_2 : (\mathbf{p}, \mathbf{z}) = \mathbf{g}(\mathbf{y}; \boldsymbol{\tau})$. Here, $\boldsymbol{\tau}$ is a fixed set of equations corresponding to

the geometric constraints, *e.g.*, the length between two joints. This is to say, $\mathbb{M}_1$ is trained to optimize $\boldsymbol{\theta}$ while $\mathbb{M}_2$ is a parameter-free algorithm which involves fitting a few fixed arithmetic equations.

To alleviate the expense of data annotation, we apply a setting known as semi-supervised learning [66] which contains two parts of training data. First, a labeled set of training data $\mathcal{D}_1 = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ is collected from the virtual environment. This process is performed automatically with little cost, and also easily transplanted to other robotic systems with a 3D model available. Second, an unlabeled set of image data $\mathcal{D}_2 = \{\tilde{\mathbf{x}}_m\}_{m=1}^{M}$ is provided, while the corresponding label $\tilde{\mathbf{y}}_m$ for each $\tilde{\mathbf{x}}_m$ remains unknown. We use $\mathcal{P}^{\mathrm{V}}$ and $\mathcal{P}^{\mathrm{R}}$ to denote the virtual and real image distributions, *i.e.*, $\mathbf{x}_n \sim \mathcal{P}^{\mathrm{V}}$ and $\tilde{\mathbf{x}}_m \sim \mathcal{P}^{\mathrm{R}}$, respectively. Since $\mathcal{P}^{\mathrm{V}}$ and $\mathcal{P}^{\mathrm{R}}$ can be different in many aspects, we cannot expect a model trained on $\mathcal{D}_1$ to generalize sufficiently well on $\mathcal{D}_2$.

The key is to bridge the gap between $\mathcal{P}^{\mathrm{V}}$ and $\mathcal{P}^{\mathrm{R}}$. One existing solution works in an *explicit* manner, which trains a mapping $\mathbf{r}(\cdot)$, so that when we sample $\tilde{\mathbf{x}}_m$ from $\mathcal{P}^{\mathrm{R}}$, $\mathbf{r}(\tilde{\mathbf{x}}_m)$ maximally mimics the distribution of $\mathcal{P}^{\mathrm{V}}$. This is achieved by unpaired image-to-image translation [67], which was verified effective in some vision tasks [55]. However, in our problem, an additional cue emerges, claiming that the source and target images have the same label distribution, *i.e.*, both scenarios aim at estimating the pose of exactly the same object, so we can make use of this cue to achieve domain adaptation in an *im-*

plicit manner. In practice, we do semi-supervised training by providing the system with unlabeled data. Our approach exhibits superior transfer ability in this specific task, while we preserve the possibility of combining both manners towards higher accuracy.

To this end, we reformulate $\mathbb{M}_1$ and $\mathbb{M}_2$ in a probabilistic style. $\mathbb{M}_1$ produces a distribution $\mathcal{F}(\mathbf{x};\boldsymbol{\theta}) \ni \mathbf{y}$, and similarly, $\mathbb{M}_2$ outputs $\mathcal{G}(\mathbf{y};\boldsymbol{\tau}) \ni (\mathbf{p},\mathbf{z})$. Here, the goal is to maximize the marginal likelihood of $(\mathbf{p},\mathbf{z})$ while $\mathbf{y}$ remains a latent variable:

$$(\mathbf{p}^\star,\mathbf{z}^\star) = \arg\max_{(\mathbf{p},\mathbf{z})} \int \Pr(\mathbf{y};\boldsymbol{\theta} \mid \mathbf{x}) \cdot \Pr(\mathbf{p},\mathbf{z};\boldsymbol{\tau} \mid \mathbf{y})\,\mathrm{d}\mathbf{y}. \tag{3.1}$$

There is another option, which directly computes $\mathbf{y}^\star = \arg\max_{\mathbf{y}} \Pr(\mathbf{y};\boldsymbol{\theta} \mid \mathbf{x})$ and then infers $\mathbf{p}^\star$ and $\mathbf{z}^\star$ from $\mathbf{y}^\star$. We do not take it because we trust $\Pr(\mathbf{p},\mathbf{z};\boldsymbol{\tau} \mid \mathbf{y})$ more than $\Pr(\mathbf{y};\boldsymbol{\theta} \mid \mathbf{x})$, since the former is formulated by strict geometric constraints. Eqn (3.1) can be solved using an iterative algorithm, starting with a model $\mathcal{F}(\mathbf{x};\boldsymbol{\theta})$ pre-trained in the virtual dataset.

In the first step, we fix $\boldsymbol{\theta}$ and infer $\mathcal{F}(\mathbf{x};\boldsymbol{\theta})$. This is done by cropping the input image to 256×256 and feeding it to a stacked hourglass network [68] with 2 stacks. The network produces $K = 17$ heatmaps, each of which, sized 64×64, corresponds to a keypoint. These heatmaps are taken as input data of $\mathcal{G}(\mathbf{y};\boldsymbol{\tau})$ which estimates $\mathbf{p}$ and $\mathbf{z}$ as well as $\mathbf{y}$. This is done by making use of geometric constraints $\boldsymbol{\tau}$, which appears as a few linear equations with fixed parameters, *e.g.*, the length of each bone of the arm. This is a probabilistic model and we

apply an iterative algorithm (see Section 3.3.5) to find an approximate solution y', p' and z'. Note that y' is not necessarily the maximum in $\mathcal{F}(\mathrm{x};\boldsymbol{\theta})$.

In the second step, we take the optimal y' to update $\boldsymbol{\theta}$. As $\mathcal{F}(\mathrm{x};\boldsymbol{\theta})$ is a deep network, this is often achieved by gradient back-propagation. We incorporate this iterative algorithm with stochastic gradient descent. In each basic unit known as an *epoch*, each step is executed only once. Although convergence is most often not achieved, we continue with the next epoch, which brings more informative supervision. Compared with solving Eqn (3.1) directly, this strategy improves the efficiency in the training stage, *i.e.*, a smaller number of iterations is required. Figure 3.3 shows an illustration of our transferable pose estimation pipeline.

3.3.4 Motion Control

In order to control the arm to complete tasks, we need a motion control module which takes the estimated 3D pose as input and outputs an action to achieve the goal. The motion control policy $a_t = \pi(s_t, g_t)$ is learned via a deep reinforcement learning algorithm. s_t is the state about the environment at time t, *e.g.*the arm pose p. g_t represents the goal, *e.g.*target location. a_t is the control signal for each joint in our system. The policy is learned in our virtual environment, and optimized by Deep Deterministic Policy Gradient (DDPG) [69]. Our experiment shows that using arm pose as input, the policy learned in the

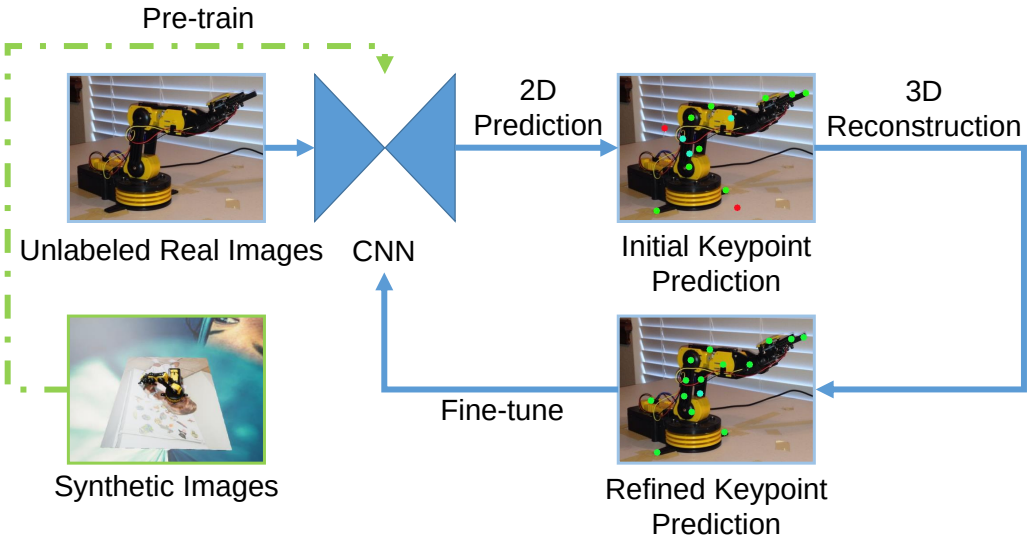

Figure 3.3: The pipeline of transferable 3D pose estimation (best viewed in color). The initial prediction may contain both accurate (green) and inaccurate (cyan) keypoints, and even outliers (red). By introducing a 3D-prior constraint, we obtain a refined keypoint prediction, which is used to fine-tune the neural network.

virtual environment can be directly applied to the real world.

3.3.5 Implementation Details

- **Training Data Variability**

Our approach involves two parts of training data, namely, a virtual subset to pre-train 2D keypoint detection, and an unlabeled real subset for fine-tuning. In both parts, we change the background contents of *each* training image so as to facilitate data variability and thus alleviate over-fitting.

In the virtual domain, background can be freely controlled by the graphical

renderer. In this scenario, we place the arm on a board, under a sky sphere, and the background of the board and the sphere are both randomly sampled from the MS-COCO dataset [70]. In the real domain, however, background parsing is non-trivial yet can be inaccurate. To prevent this difficulty, we create a special subset for fine-tuning, in which all images are captured in a clean environment, *e.g.*, in front of a white board, which makes it easy to segment the arm with a pixel-wise color-based filter, and then place it onto a random image from the MS-COCO dataset. We observe consistent accuracy gain brought by these simple techniques.

- **Joint Keypoint Detection and Pose Estimation**

We use an approximate algorithm to find the $\mathbf{y}'$ (as well as $\mathbf{p}'$ and $\mathbf{z}'$) that maximizes $\Pr(\mathbf{y}; \boldsymbol{\theta} \mid \mathbf{x}) \cdot \Pr(\mathbf{p}, \mathbf{z}; \boldsymbol{\tau} \mid \mathbf{y})$ in Eqn (3.1), because an accurate optimization is mathematically intractable. We first compute $\mathbf{y}' = \arg\max_{\mathbf{y}} \mathcal{F}(\mathbf{x}; \boldsymbol{\theta})$ which maximizes $\Pr(\mathbf{y}; \boldsymbol{\theta} \mid \mathbf{x})$. This is performed on the heatmap of each 2D keypoint individually, which produces not only the most probable $\mathbf{y}'_k$ but also a score c_k indicating its confidence. We first filter out all keypoints with a threshold ξ, *i.e.*, all keypoints with $c_k < \xi$ are considered unknown (and thus completely determined by geometric prior) in the following 3D reconstruction module. This is to maximally prevent the impact of outliers. In practice, we use $\xi = 0.3$ and our algorithm is not sensitive to this parameter.

Next, we recover the 3D pose using these 2D keypoints, *i.e.*, maximizing

$\Pr(\mathbf{p}, \mathbf{z}; \boldsymbol{\tau} \mid \mathbf{y} = \mathbf{y}')$. Under the assumption of perspective projection, that each keypoint $\mathbf{y}_k \in \mathbb{R}^2$ is the 2D projection of a 3D coordinate $\mathbf{z}_k \in \mathbb{R}^3$, which can be written in a linear equation:

$$[\mathbf{y}|\mathbf{1}]^\top \cdot \hat{\mathbf{S}} = \mathbf{K} \cdot [\mathbf{R}|\mathbf{T}] \cdot [\mathbf{z}|\mathbf{1}]^\top. \tag{3.2}$$

Here, $\mathbf{y} \in \mathbb{R}^{K\times 2}$ and $\mathbf{z} \in \mathbb{R}^{K\times 3}$ are 2D and 3D coordinate matrices, respectively, and $\mathbf{1} \in \mathbb{R}^{K\times 1}$ is an all-one vector. $\mathbf{K} \in \mathbb{R}^{3\times 3}$ is the camera intrinsic matrix, which is constant for a specific camera. $\mathbf{S} \in \mathbb{R}^K$, $\mathbf{R} \in \mathbb{R}^{3\times 3}$ and $\mathbf{T} \in \mathbb{R}^{3\times 1}$ denote the scaling vector, rotation matrix and translation vector, respectively, all of which are determined by $\mathbf{p}$. For each keypoint k, $\mathbf{z}^k$ is determined by the motor transformation $\mathbf{z}^k = \mathbf{z}_0^k \cdot \mathbf{W}^k$, where $\mathbf{z}_0 \in \mathbb{R}^{K\times 3}$ is a constant matrix indicating the coordinates of all keypoints when motor angles are 0, and $\mathbf{W}^k \in \mathbb{R}^{3\times 3}$ is the motor transformation matrix for the kth keypoint, which is also determined by $\mathbf{p}$. $\hat{\mathbf{S}} = \mathrm{diag}(\mathbf{S})$ is the scaling matrix. Due to the inaccuracy in prediction ($\mathbf{y}$ can be inaccurate in either prediction or manual annotation) and formulation (*e.g.*, perspective projection does not model camera distortion), Eqn (3.2) may not hold perfectly. In practice, we assume the recovered 3D coordinates to follow an isotropic Gaussian distribution, and so maximizing its likelihood gives the

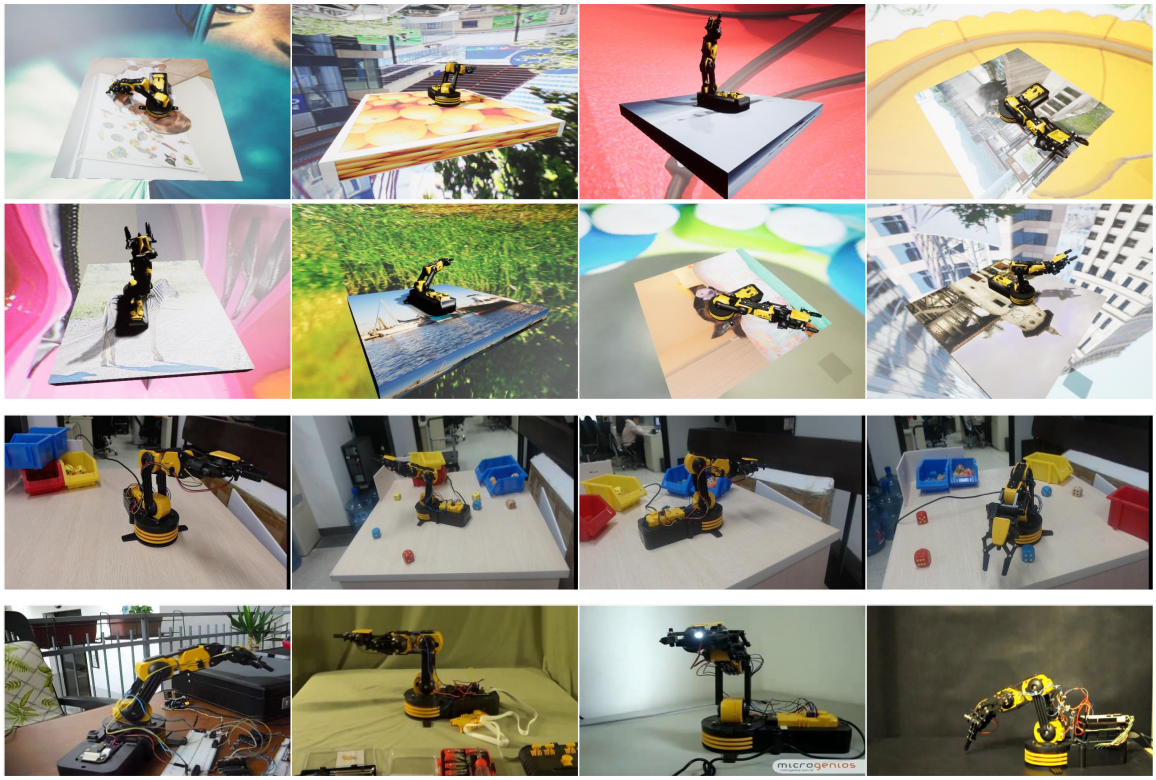

Figure 3.4: Example images of three datasets used in this chapter. From top to bottom: synthetic images (top two rows), lab images and YouTube images. Please zoom in to see details.

following log-likelihood loss:

$$\mathcal{L}(\mathbf{p}, \mathbf{z} \mid \mathbf{y}) = \left\| [\mathbf{y}|\mathbf{1}]^\top \cdot \hat{\mathbf{S}} - \mathbf{K} \cdot [\mathbf{R}|\mathbf{T}] \cdot [\mathbf{z}|\mathbf{1}]^\top \right\|_2^2. \tag{3.3}$$

3.4 Experiments

3.4.1 Dataset and Settings

We generated $5{,}000$ synthetic images with randomized camera parameters, lighting conditions, arm poses and background augmentation (see Section 3.3.5). Among them, $4{,}500$ are used for training and the remaining 500 for validation. This dataset, later referred to as the **virtual dataset**, is used to verify the 2D keypoint detection model, *e.g.*, a stacked hourglass network, works well.

In the real environments, we collected and manually annotated two sets of data. The first one is named the **lab dataset**, which contains more than $20{,}000$ frames captured by a 720P Webcam. We manually chose 428 key frames and annotated them. For this purpose, we rendered the 3D model of the arm in the virtual environment and adjusted it to the same pose of the real arm so that arms in these two images exactly overlap with each other – in this way we obtained the ground-truth arm pose, as well as the camera intrinsic and extrinsic (obtained by a checkerboard placed alongside the robotic arm at the beginning of each video) parameters. We deliberately put distractors, e.g. colorful boxes, dices and balls, to make the dataset more difficult and thus evaluate the generalization ability of our models. The frames used for fine-tuning and for testing come from different videos.

The second part of real-world image data, the **YouTube dataset**, is crawled

from YouTube, which contains 109 videos with the OWI-535 arm. This is a largely diversified collection, in which the arm may even be modded, *i.e.*, the geometric constraints may not hold perfectly. We sampled 275 frames and manually annotated the visibility as well as position for each 2D keypoint. Note that, without camera parameters, we cannot annotate the accurate pose of the arm. This dataset is never included in training, but used to observe the behavior of domain adaptation.

Sample images of three datasets are shown in Figure 3.4.

3.4.2 Pose Estimation

3.4.2.1 Detecting 2D Keypoints

We first evaluate 2D keypoint detection, and make use of a popular metric named PCK@0.2 [71] to evaluate the accuracy. For this purpose, we train a 2-stack hourglass network from scratch for 30 epochs in the **virtual** dataset. Standard data augmentation techniques are applied, including random translation, rotation, scaling, color shifting and flipping. On top of this model, we consider several approaches to achieve domain transfer. One is to train an explicit model which transfers virtual data to fake real data on which we train a new model. In practice, we apply a popular generative model named CycleGAN [67]. We trained the CycleGAN network with synthetic image as the

Model	Virtual	Lab	YouTube	YouTube-vis
Synthetic	**99.95**	95.66	80.05	81.61
CycleGAN	99.86	97.84	75.26	76.98
ADDA	99.89	96.14	79.19	80.04
CyCADA	99.84	98.09	73.47	74.37
Our Method	99.63	**99.55**	**87.01**	**88.89**

Table 3.1: 2D keypoint detection accuracy (PCK@0.2, %) on three datasets. Models are tested on YouTube dataset when considering all keypoints and considering only the visible ones.

source domain and lab image as the target domain for 100 epochs. Other domain adaptation methods, *i.e.*, ADDA [58] and its follow-up work CyCADA [55] are also applied and compared with our approach described in Section 3.3.3. We mix the synthetic and real images with a ratio of $6:4$ and use the same hyper-parameters as for the baseline. Background clutters are added to the lab images in an online manner to facilitate variability (see Section 3.3.5).

Results are summarized in Table 3.1. The baseline model works almost perfectly on **virtual** data, which reports a PCK@0.2 accuracy of 99.95%. However, this number drops significantly to 95.66% in lab data, and even dramatically to 80.05% in **YouTube** data, demonstrating the existence of domain gaps. These gaps are largely shrunk after domain adaptation algorithms are applied. Training with images generated by CycleGAN, we found that the model works better in its target domain, i.e. the **lab** dataset by a margin of 2.18%. However, this model failed to generalize to **YouTube** dataset, as the accuracy is even lower than the baseline model. The cases are similar for ADDA and CyCADA,

which gains 0.48% and 2.43% improvement on the **lab** dataset respectively, but both of the approaches do not generalize well to the **YouTube** dataset. Our approach, on the other hand, achieves much higher accuracy, with a PCK@0.2 score of 99.55% in the **lab** data, and 87.01% in the **YouTube**, boosting the baseline performance by 6.96%. In the subset of visible **YouTube** keypoints, the improvement is even higher (7.28%). In addition, the refined model only produces a slightly worse PCK@0.2 accuracy (99.63% vs. 99.95%) on **virtual** data, implying that balance is achieved between "fitting on virtual data" and "transferring to real data".

The results reveal that performance of explicit domain adaptation manners, *i.e.* CycleGAN, ADDA and CyCADA can be limited in several aspect. For instance, compared with our 3D geometric based domain adaptation method, although the models trained with these explicit domain adaptation methods fit to the target domain, it has a poor performance on unseen data. Moreover, we fail to train a CycleGAN model with **YouTube** dataset as the target domain, because the distribution of data in YouTube is too diverse and such transformation is hard to learn.

3.4.2.2 Estimating the 3D Pose

We first test the performance of 3D pose estimation on the **virtual** dataset. We use the model trained only on synthetic images since it has the best 2D

Model	Motor					Camera	
	Rotation	Base	Elbow	Wrist	Average	Rotation	Location
Synthetic	7.41	6.20	7.15	7.74	7.13	6.29	7.58
Refined	**4.40**	**3.29**	**5.35**	**6.03**	**4.81**	**5.29**	**6.73**

Table 3.2: 3D pose estimation errors (degrees) and camera parameter prediction errors (degrees and centimeters) in the **lab** dataset.

Figure 3.5: Qualitative results from our YouTube dataset. The challenges include occlusion, user modification, lighting, etc. We show synthetic images generated using the camera parameters and pose estimated from the single input image. Both success cases (left five columns) and failure cases (rightmost column) are shown.

prediction accuracy on synthetic data. The experiment was conducted on 500 synthetic images. The angular error for four joints are $2.67°$, $2.80°$, $2.76°$, $3.31°$, with an average of $2.89°$. The error of camera parameters is $2.25°$ for rotation and 2.59cm for location.

We also test the 3D pose estimation performance of our model on real images, which is the basis for completing tasks. The quantitative 3D pose estimation result is only reported for the **lab** dataset, since getting 3D annotation for **YouTube** data is difficult. We estimate the camera intrinsic parameters by using camera calibration with checkerboard. Results are shown in Table 3.2. The results reveal that our refined model outperforms the synthetic only model by $2.32°$ on average angular error. The qualitative result on **YouTube** dataset are shown in Figure 3.5. Since the camera intrinsic parameters are unknown for YouTube videos, we use the weak perspective model during reconstruction. Heavy occlusion, user modification and extreme lighting make the 3D pose estimation hard in some cases. We select typical samples for success and failure cases.

3.4.2.3 Ablation Study: Domain Adaptation Options

As described in Section 3.3.5, when training the refined model, two strategies are applied on the real images: random background augmentation and joint keypoint detection and pose estimation. To evaluate the contribution of

Model	Lab	YouTube	YouTube-vis
Synthetic Only	95.66	80.05	81.61
BG ✗ 3D ✗	94.52	80.24	82.22
BG ✓ 3D ✗	98.72	84.04	87.11
BG ✗ 3D ✓	97.31	86.52	88.27
BG ✓ 3D ✓	**99.55**	**87.01**	**88.89**

Table 3.3: 2D keypoint detection accuracy (PCK@0.2, %) under ablation study. '3D' stands for joint keypoint detection and 3D pose estimation, and 'BG' for random background augmentation.

these two strategies to the improvement on accuracy, we did an ablation study. Results are shown in Table 3.3.

We compare the performance of 5 models: 1) baseline model, trained on $4{,}500$ synthetic images; 2) - 5) models trained with/without joint estimation and with/without background augmentation. Note that if a model is trained without joint estimation, we directly take the argmax on the predicted heatmap as annotations for training. We see that both strategies contribute to the overall performance improvement. Our model performs best when combining both strategies.

3.4.2.4 Ablation Study: Number of Training Images

With the help of domain randomization, we can generate an unlimited number of synthetic images with high abundance in their appearance. On the other hand, the performance of deep models tends to saturate as the number of training data increases. Therefore, it is necessary to balance between performance

and data-efficiency.

Results are shown in Table 3.4. As the number of training images increases from $2{,}500$ to $5{,}000$, the accuracy significantly increases (by 3.28% and 4.81% on **lab** and **YouTube** dataset, respectively). When the number of training images continue to increase to $20{,}000$, the accuracy only increases by a small margin (1.49% and 0.22% on **lab** and **YouTube** dataset, respectively). Therefore, $5{,}000$ synthetic images is a nice balance between accuracy and efficiency.

# Images	Lab	YouTube	YouTube-vis
2,500	92.38	75.24	77.33
5,000	95.66	80.05	81.61
10,000	96.13	80.71	81.71
20,000	97.25	81.11	81.83

Table 3.4: 2D keypoint detection accuracy (PCK@0.2, %) with respect to the number of training images.

3.4.3 Controlling the Arm with Vision

We implement a complete control system purely based on vision, as described in Section 3.3.4. It takes a video stream as input, estimates the arm pose, then plans the motion and sends control signal to the motors.

This system is verified with a task, reaching a target point. The goal is controlling the arm to make the arm tip reach right above a specified point on the table without collision. Each attempt is considered successful if the horizontal distance between arm tip and the target is within 3cm. The system

is tested at 6 different camera views. For each view, the arm needs to reach 9 target points. The target points and camera views are selected to cover a variety of cases. We also place distractors to challenge our vision module. A snapshot of our experiment setup can be seen in Figure 3.6.

We report human performance on the same task. Human is asked to watch the video stream from a screen and control the arm with a game pad. This setup ensures human and our system accepts the same vision input in comparison. In addition, we allow human to directly look at the arm and move freely to observe the arm when doing the task. The performance for both setups are reported.

Our control system can achieve comparable performance with human in this task. The result is reported in Table 3.5. Human can perform much better if directly looking at the arm. This is because human can constantly move his head to pick the best view for current state. Potentially, we could use action vision, or multi-camera system, to mimic this ability and further improve the system, which are interesting future work but beyond the scope of this work. It is worth noticing our system can run real-time and finish the task faster than human.

Built on this control module, we show that our system can move a stack of dices into a box separately. Please see https://youtu.be/8hZjdqDrYas for video demonstration. To simplify deployment, we directly feed the ground-truth lo-

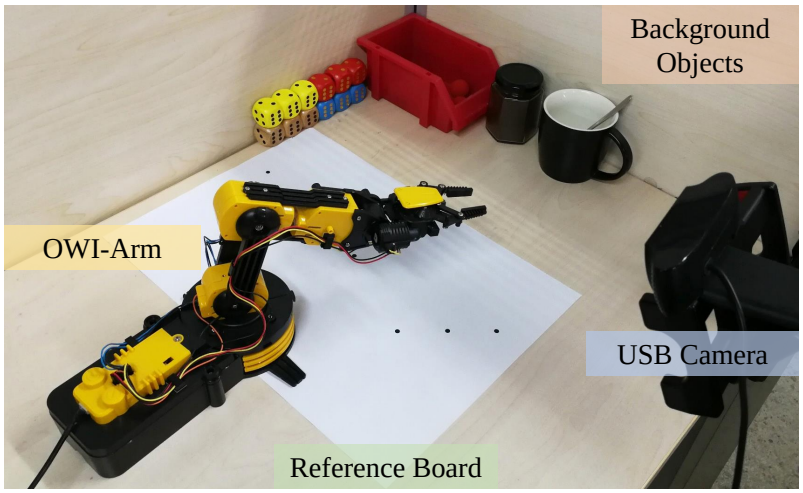

Figure 3.6: A snapshot of the real-world experiment setup. Locations of the goals are printed on the reference board and are used as reference when measuring the error. We scatter background objects randomly during testing.

cations of the dices and the box into the system and apply the same controller as the reaching task. The successful rate of this task is not high, because it requires highly accurate pose control in both horizontal and vertical directions. We also provide some interesting failure cases in the video. Failure cases are caused by several reasons, *e.g.*, self occlusion or rarely seen configurations. Also, the controller sometimes fails, *e.g.*, if the arm is too far from the camera, a long-distance movement only causes minor visual difference. At the current point, this demo reveals the potential of our vision-based control system, as well as advocates more advanced vision algorithms to be designed to improve its performance.

Agent	Input Type	Distance Error (cm)	Success Rate	Average Time (s)
Human	Direct	0.65	100.0%	29.8
Human	Camera	2.67	**66.7%**	38.8
Ours	Camera	**2.66**	59.3%	**21.2**

Table 3.5: Quantitative result for completing the reaching task. Our system achieves comparable performance with human when the same input signal is given.

3.5 Conclusion

In this chapter, we built a system, which is purely based on vision inputs, to control a low-cost, sensor-free robotic arm to accomplish tasks. We used a semi-supervised algorithm which integrates labeled synthetic as well as unlabeled real data to train the pose estimation module. Geometric constraints of multi-rigid-body system (the robotic arm in this case) was utilized for domain adaptation. Our approach, with merely a 3D model being required, has the potential to be applied to other multi-rigid-body systems.

To facilitate reproducible research, we created a virtual environment to generate synthetic data, and also collected two real-world datasets from our lab and YouTube videos, respectively, all of which can be used as benchmarks to evaluate 2D keypoint detection and/or 3D pose estimation algorithms. In addition, the low cost of our system enables vision researchers to study robotic tasks, *e.g.*, reinforcement learning, imitation learning, active vision, *etc.*, without large economic expenses. This system also has the potential to be used for

high-school and college educational purposes.

Beyond our work, many interesting future directions can be explored. For example, we can train perception and controller modules in a joint, end-to-end manner [12] [36], or incorporate other vision components, such as object detection and 6D pose estimation, to enhance the ability of the arm so that more complex tasks can be accomplished.

Chapter 4

Consistency-Constrained Semi-Supervised Learning for Synthetic Animals

In this chapter, we present a method to train an animal keypoint detection algorithm trained with synthetic data. Researchers and industry spent enormous resources annotating human keypoints. But it is difficult to do the same for all animal species. Instead, we use synthetic images and generated ground truth (keypoint) to train keypoint detectors for animals. The appearance diversity of synthetic animals is limited compared with real animal images. This leads to a big domain gap between real and synthetic. In this case, the geometry constraint proposed in Chapter 3 is no longer valid, since the 3D ge-

ometry between real and synthetic animals is different due to shape and pose variation. In order to overcome this domain challenge, we design a consistency-constrained semi-supervised learning method and combine it with domain randomization. This method enables us to train a robust keypoint detector using synthetic images.

4.1 Introduction

Thanks to the presence of large scale annotated datasets and powerful Convolutional Neural Networks(CNNs), the state of human parsing has advanced rapidly. By contrast, there is little previous work on parsing animals. Parsing animals is important for many tasks, including, but not limited to monitoring wild animal behaviors, developing bio-inspired robots, building motion capture systems, etc.

One main problem for parsing animals is the limit of datasets. Though many datasets containing animals are built for classification, bounding box detection, and instance segmentation, only a small number of datasets are built for parsing animal keypoints and parts. Annotating large scale datasets for animals is prohibitively expensive. Therefore, most existing approaches for parsing humans, which often require enormous annotated data [72] [73], are less suited for parsing animals.

In this chapter, we use synthetic data to address this challenge. Many works [54] [74] show that by jointly using synthetic images and real images, models can yield supreme results. In addition, synthetic data also has many unique advantages compared to real-world datasets. First, rendering synthetic data with rich ground truth at scale is easier and cheaper compared with capturing and annotating real-world images. Second, synthetic data can also provide accurate ground truth for cases where annotations are hard to acquire for natural images, such as labeling optical flow [15] or under occlusion and low-resolution. Third, real-world datasets usually suffer from the long-tail problem where rare cases are less represented. Generated synthetic datasets can avoid this problem by sampling rendering parameters.

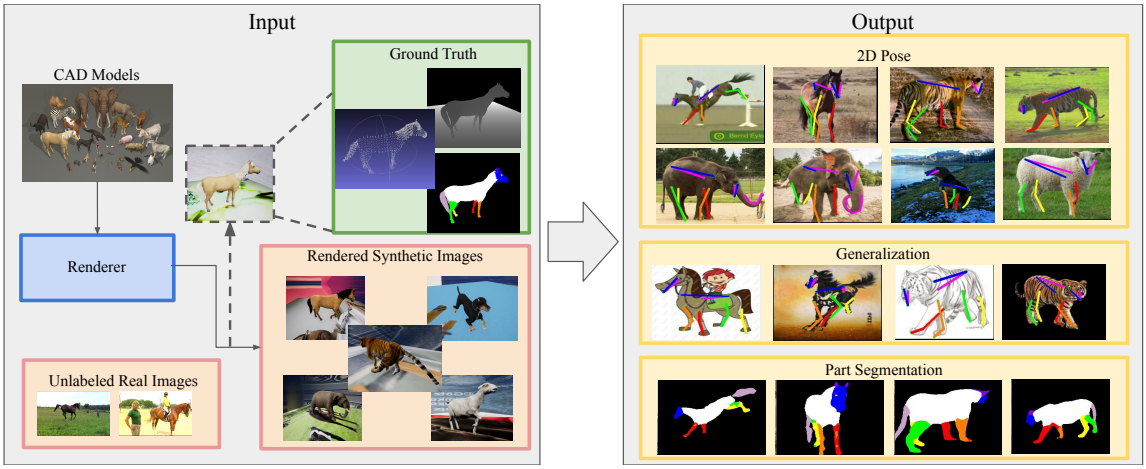

Figure 4.1: Overview. We generate a synthetic animal dataset by randomly sampling rendering parameters including camera viewpoints, lighting, textures and poses. The dataset contains 10+ animals along with rich ground truth, such as dense 2D keypoints, part segmentation and depth maps. With the synthetic dataset, we propose an effective method which allows for accurate keypoint prediction across domains. In addition to 2D pose estimation, we also show models can predict accurate part segmentation.

However, there are large domain gaps [75] [5] [55] between synthetic images and real images, which prevent models trained on synthetic data from generalizing well to real-world images. Moreover, synthetic data is also limited by object diversity. ShapeNet [13] has been created to include diverse 3D models and SMPL [4] has been built for humans. Nevertheless, creating such diverse synthetic models is a difficult task, which requires capturing the appearance and attaching a skeleton to the object. Besides, considering the number of animal categories in the world, creating diverse synthetic models along with realistic textures for each animal is almost infeasible.

In this chapter, we propose a method where models are trained using synthetic CAD models. Our method can achieve high performance with only a single CAD animal model. We generate pseudo-labels on unlabeled real images for semi-supervised learning. To handle noisy pseudo-labels, we design three consistency-check criteria to evaluate the quality of the predicted labels, which we refer to as consistency-constrained semi-supervised learning (CC-SSL). Through extensive experiments, we show that our models achieve similar performance to models trained on real data, but without using any annotation of real images. It also outperforms other domain adaptation methods by a large margin. Providing real image annotations, the performance can be further improved. Furthermore, we demonstrate models trained with synthetic data show better domain generalization performance in multiple visual

domains compared with those trained on real data.

We summarize the contributions of this chapter as follows. First, we propose a consistency-constrained semi-supervised learning framework (CC-SSL) to learn a model with one single CAD object. We show that models trained with synthetic data and unlabeled real images allow for accurate keypoint prediction on real images. Second, when using real image labels, we show that models trained jointly on synthetic and real images achieve better results compared to models trained only on real images. Third, we evaluate the generalizability of our learned models across different visual domains in the Visual Domain Adaptation Challenge dataset and we quantitatively demonstrate that models trained using synthetic data show better generalization performance than models trained on real images. Lastly, we generate an animal dataset with 10+ different animal CAD models and we demonstrate the data can be effectively used for 2D pose estimation, part segmentation, and multi-task learning.

4.2 Related Work

4.2.1 Animal Parsing

Though there exists large scale datasets containing animals for classification, detection, and instance segmentation, there are only a small number of

datasets built for pose estimation [76] [77] [78] [79] [80] and animal part segmentation [81]. Besides, annotating keypoints or parts is time-consuming and these datasets only cover a tiny portion of animal species in the world.

Due to the lack of annotations, synthetic data has been widely used to address the problem [82] [83] [65] [84]. Similar to SMPL models [4] for humans, [84] proposes a method to learn articulated SMAL shape models for animals. Later, [65] extracts more 3D shape details and is able to model new species. Unfortunately, these methods are built on manually extracted silhouettes and keypoint annotations. Recently, [82] proposes to copy texture from real animals and predicts 3D mesh of animals in an end-to-end manner. Most related to our method is [83], where authors propose a method to estimate animal poses on real images using synthetic silhouettes. Different from [83] which requires an additional robust segmentation model for real images during inference, our strategy does not require any additional models.

4.2.2 Unsupervised Domain Adaptation

Unsupervised domain adaptation focuses on learning a model that works well on a target domain when provided with labeled source samples and unlabeled target samples. A number of image-to-image translation methods [85] [67] [86] are proposed to transfer images from different domains. Another line of work studies how to explicitly minimize some measure of feature difference,

such as maximum mean discrepancy [87] [88] or correlation distances [89] [90]. [91] proposes to explicitly partition features into a shared space and a private space. Recently, adversarial loss [58] [55] is used to learn domain invariant features, where a domain classifier is trained to distinguish the source and target distributions. [58] proposes a general framework to bring features from different domains closer. [55] [92] extend this idea with cycle consistency to improve results.

Recent works have also investigated how to use these techniques to advance deformable objects parsing. [75] studies using synthetic human images combined with domain adaptation to improve human 3D pose estimation. [5] renders 145 realistic synthetic human models to reduce the domain gap. Different from previous works where a large amount of realistic synthetic models are required, we show that models trained on one CAD model can learn domain-invariant features.

4.2.3 Self-training

Self-training has been proved effective in semi-supervised learning. Early work [93] draws the connection between deep self-training and entropy regularization. However, since generated pseudo-labels are noisy, a number of methods [94] [95] [96] [97] [98] [99] [100] [101] [102] [103] are proposed to address the problem. [96] [97] formulate self-training as a general EM algorithm

and proposes a confidence regularized self-training framework. [98] proposes a self-ensembling framework to bootstrap models using unlabeled data. [99] extends the previous work to unsupervised domain adaptation and demonstrate its effectiveness in bridging domain gaps.

Closely related to our work on 2D pose estimation is [102], where the authors propose a simple method for omni-supervised learning that distills knowledge from unlabeled data and demonstrate its effectiveness on detection and pose estimation. However, under large domain discrepancy, the assumption that the teacher model assigns high-confidence pseudo-labels is not guaranteed. To tackle the problem, we introduce a curriculum learning strategy [104] [105] [106] to progressively increase pseudo-labels and train models in iterations. We also extend [102] by leveraging both spatial and temporal consistencies.

4.3 Approach

We first formulate a unified image generation procedure in Section 4.3.1 built on the low dimension manifold assumption. In Section 4.3.2, we define three consistencies and discuss how to take advantage of these consistencies during pseudo-label generation process. In Section 4.3.3, we propose a Pseudo-Label Generation algorithm using consistency-check. Then in Section 4.3.4 we

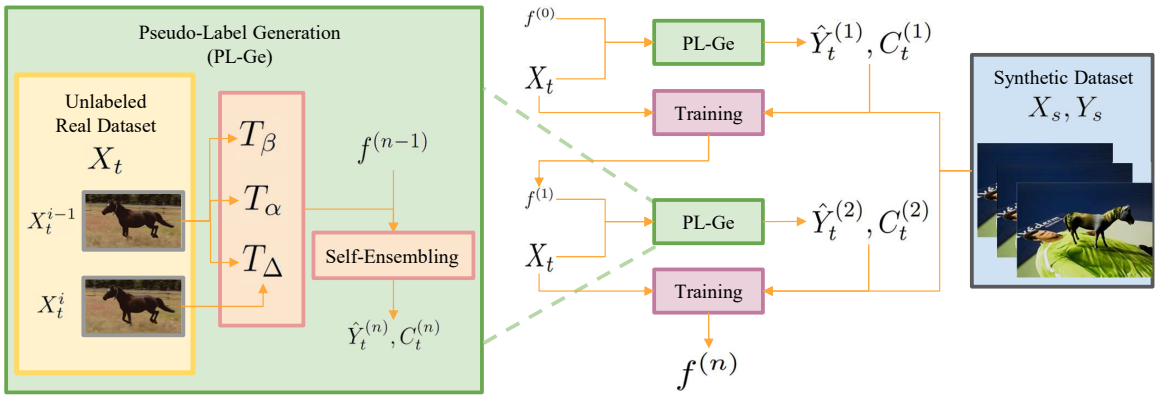

Figure 4.2: Consistency-constrained semi-supervised learning pipeline. T_β indicates the invariance consistency, T_α indicates the equivariance consistency and T_Δ indicates the temporal consistency. The training procedure can be described as follows: we start with training a model only using synthetic data and obtain an initial model $f^{(0)}$. Then we iterate the following procedure. For the nth iteration, we first use the proposed Pseudo-Label Generation Algorithm 1 to generate labels $\hat{Y}_t^{(n)}$. Next, we train the model using (X_s, Y_s) and $(X_t, \hat{Y}_t^{(n)})$ jointly.

present our consistency-constrained semi-supervised learning algorithm and discuss the iterative training pipeline. Lastly, in Section 4.3.5, we explain how our synthetic datasets are generated.

We consider the problem under unsupervised domain adaptation framework with two datasets. We name our synthetic dataset as the source dataset (X_s, Y_s) and real images as the target dataset X_t. The goal is to learn a model f to predict labels for the target data X_t. We simply start with learning a source model f_s using paired data (X_s, Y_s) in a fully supervised way. Then we bootstrap the source model using target dataset with consistency-constrained semi-supervised learning. An overview of the pipeline is presented in Figure 4.2.

4.3.1 Formulate Image Generation Procedure

In order to learn a model using synthetic data that can generalize well to real data, one needs to assume that there exists some essential knowledge shared between these two domains. Take animal 2D pose estimation as an example, though synthetic and natural images look differently by textures and background, they are quite similar in terms of poses and shape. Actually, these are exactly what we hope a model trained on synthetic data can learn. So an ideal model should be able to capture these essential factors and ignore those less relevant ones, such as lighting and background.

Formally, we introduce a generator G that transforms poses, shapes, viewpoints, textures, etc, into an image. Mathematically, we group all these factors into two categories, task-related factors α, which is what a model cares about, and others β, which are irrelevant to the task at hand. So we parametrize the image generation process as follows,

$$X = G(\alpha, \beta) \tag{4.1}$$

where X is a generated image and G denotes the generator. Specifically, for 2D pose estimation, α represents factors related to the 2D keypoints, such as pose and shape; β indicates factors independent of α, which could be textures, lighting and background.

4.3.2 Consistency

Based on the formulation in Section 4.3.1, we define three consistencies and discuss how to take advantage of these consistencies for the pseudo-label generation process.

Since model-generated labels on the target dataset are noisy, one needs to tell the model which predictions are correct and which are wrong. Intuitively, an ideal 2D keypoint detector should generate consistent predictions on one image no matter how the background is perturbed. In addition, if one rotates the image, the prediction should change accordingly as well. Based on these intuitions, we propose to use consistency-check to reduce false positives.

In the following paragraphs, we will introduce invariance consistency, equivariance consistency and temporal consistency. We will discuss how to use consistency-check to generate pseudo-labels, which serves as the basis for the proposed semi-supervised learning method.

The transformation applied to an image can be considered as directly transforming the underlying factors in Equation 4.1. We define a general tensor operator, $T : \mathbb{R}^{H\times W} \rightarrow \mathbb{R}^{H\times W}$. In addition, we introduce τ_α corresponding to operations that would affect α and τ_β to represent operations independent of α. Then Equation 4.1 can be expressed as following,

$$T(X) = G(\tau_\alpha(\alpha), \tau_\beta(\beta)) \tag{4.2}$$

We use $f : \mathbb{R}^{H \times W} \rightarrow \mathbb{R}^{H \times W}$ to denote a perfect 2D pose estimation model. When f is applied to Equation 4.2, it is obvious that, $f[T(X)] = f[G(\tau_\alpha(\alpha), \tau_\beta(\beta))]$.

Invariance consistency: If the transform T does not change factors associated with the task, the model's prediction is expected to be the same. The idea here is that a well-behaved model should be invariant to operations on β. For example, in 2D pose estimation, adding noise to the image or perturbing colors should not affect the model's prediction. We name these transforms *invariance transform* T_β, as shown in Equation 4.3.

$$f[T_\beta(X)] = f(X) \tag{4.3}$$

If we apply multiple invariance transforms to the same image, the predictions on these transformed images should be consistent. This consistency can be used to verify whether the prediction is correct, which we refer to as *invariance consistency*.

Equivariance consistency: Besides invariance transform, there are other cases where the task related factors are changed. We use T_α to denote transforms related to operations τ_α. There are special cases where we can easily get the corresponding T_α. One easy case is that, sometimes, the effect of τ_α only cause geometric transformations in 2D images, which we refer to as *equivari-*

ance transform T_α. Actually, this is essentially similar to what [102] proposes. Therefore, we have *equivariance consistency* as shown in Equation 4.4.

$$f[T_\alpha(X)] = T_\alpha[f(X)] \tag{4.4}$$

It is also easy to show that $f(X) = T_\alpha^{-1}[f[T_\alpha(X)]]$, which means that, after applying the inverse transform T_α^{-1}, a good model should give back the original prediction.

Temporal consistency: It is difficult to model transformations between frames in a video. This transform T_Δ does not satisfy the invariance and equivariance properties described above. However, T_Δ is still induced by variations of underlying factors α and β. It is reasonable to assume that, in a real-world video, these factors do not change dramatically between neighboring frames.

$$f[T_\Delta(X)] = f(X) + \Delta \tag{4.5}$$

So we assume the keypoints shifting between two frames is relatively small as shown in Equation 4.5. Intuitively, this means that the keypoint prediction for the same joint in consecutive frames should not be too far away, otherwise it is likely to be incorrect.

For 2D keypoint estimation, we observe that T_Δ can be approximated by

optical flow to get Δ, which allows us to use optical flow to propagate pseudo-labels from confident frames to less confident ones.

Although we define these three consistencies for 2D pose estimation, they can be easily extended to other problems. For example, in 3D pose estimation, α can be factors related to 3D pose. Then the invariance consistency is still the same, but the equivariance consistency no longer holds, since the mapping of 3D pose to 2D pose is not a one-to-one mapping and there are ambiguities in the depth dimension. However, one can still use it as a constraint for the other two dimensions, which means the projected poses should still satisfy the same consistency. So it is easy to see that though corresponding consistencies may change for different tasks, they all follow the same philosophy.

4.3.3 Pseudo-Label Generation

In this section, we explain in details how to apply these consistencies in practice for generating pseudo-labels and propose the pseudo-label generation method as in Algorithm 1.

We address the noisy label problem in two ways. First, we develop an algorithm to generate pseudo-labels using consistency-check to remove false positives, assuming that labels generated using the correct information always satisfy these consistencies. Second, we apply the curriculum learning idea to gradually increase the number of training samples and learn models in an it-

Algorithm 1 Pseudo-Label Generation Algorithm

Input: Target dataset X_t; model $f^{(n-1)}$; decay factor λ_{decay}.

Intermediate Result: P_β, P_α are predictions after applying invariance and equivariance transform.

Output: Pseudo-labels $\hat{Y}_t^{(n)}$; confidence score $C_t^{(n)}$.

for X_t^i in X_t **do**

▷ *Invariance Consistency*

$P_\beta = f^{(n-1)}(T_\beta(X_t^i))$

▷ *Equivariance Consistency*

$P_\alpha = T_\alpha^{-1}[f^{(n-1)}(T_\alpha(X_t^i))]$

▷ *Self-Ensembling*

Ensemble P_β and P_α to get ($\hat{Y}_t^{(n),i}$, $C_t^{(n),i}$)

▷ *Temporal Consistency*

if $C_t^{(n),i} / C_t^{(n),i-1} < \lambda_{decay}$ **then**

$\hat{Y}_t^{(n),i} = (\hat{Y}_t^{(n),i-1}) + \Delta$

$C_t^{(n),i} = \lambda_{decay} * C_t^{(n),i-1}$

end if

end for

Sort $C_t^{(n)}$ and obtain C_{thresh} based on a fixed curriculum learning policy.

Set $C_t^{(n),i} = \mathbb{1}(C_t^{(n),i} \geq C_{thresh}), \quad \forall i$

erative fashion.

For the nth iteration, with the previous model $f^{(n-1)}$ obtained from the $(n-1)$th iteration, we iterate through each image X_t^i in the target dataset X_t. $f^{(n-1)}$ is not updated in this process. First, for each image, we apply multiple invariance transform T_β, equivariance transform T_α to X_t^i, and ensemble all predictions P_β and P_α to get a pair of estimated labels and confidence scores ($\hat{Y}_t^{(n),i}$, $C_t^{(n),i}$).

Second, we use temporal consistency to update weak predictions. For each keypoint, we check whether the current confidence score $C_t^{(n),i}$ is strong compared to the one in the previous frame $C_t^{(n),i-1}$ with respect to a decay factor λ_{decay}. If the current frame prediction is confident, we simply keep it; otherwise, we replace the prediction $\hat{Y}_t^{(n),i}$ with the flow prediction Δ plus the previous frame prediction and replace $C_t^{(n),i}$ with previous frame confidence multiplied by a decay factor λ_{decay}. Temporal consistency is optional and can be used if videos are available.

To this end, the algorithm has generated labels and confidence scores for all images. The last step is to iterate through the target dataset again to select C_{thresh} using a curriculum learning strategy, which determines the percentage of labels used for training. The idea here is to use keypoints with high confidence first and gradually include more keypoints after iterations. In practice, we use a policy to include top 20% ranking keypoints at the beginning, 40% for

the second iteration, until hitting 80%.

4.3.4 Consistency-Constrained Semi-Supervised Learning (CC-SSL)

For the nth iteration, the loss function $L^{(n)}$ is defined to be the Mean Square Error on heatmaps of both the source data and target data, as in Equation 4.6. γ is used to balance the loss between source and target datasets.

$$\begin{aligned} L^{(n)} = & \sum_i L_{MSE}(f^{(n)}(X_s^i), Y_s^i) \\ & + \gamma \sum_j L_{MSE}(f^{(n)}(X_t^j), \hat{Y}_t^{(n-1),j}) \end{aligned} \tag{4.6}$$

To this end, we present our Consistency-Constrained Semi-Supervised Learning (CC-SSL) approach as follows: we start with training a model only using synthetic data and obtain an initial weak model $f^{(0)} = f_s$. Then we iterate the following procedure. For the nth iteration, we first use Algorithm 1 to generate labels $\hat{Y}_t^{(n)}$. With the generated labels, we simply train the model using (X_s, Y_s) and $(X_t, \hat{Y}_t^{(n)})$ jointly using $L^{(n)}$.

4.3.5 Synthetic Dataset Generation

In order to create a diverse combination of animal appearances and poses, we collect a synthetic animal dataset containing 10+ animals. Each animal comes with several animation sequences. We use Unreal Engine to collect rich ground truth and enable nuisance factor control. The implemented factor control includes randomizing lighting, textures, changing viewpoints and animal poses.

The pipeline for generating synthetic data is as follows. Given a CAD model along with a few animation sequences, an animal with random poses and random texture is rendered from a random viewpoint for some random lighting and a random background image. We also generate ground truth depth maps, part segmentation and dense joint locations (both 2D and 3D). See Figure 4.1 for samples from the synthetic dataset.

4.4 Experiments

First, we quantitatively test our approach on the TigDog dataset [76] in Section 4.4.2. We compare our method with other popular unsupervised domain adaptation methods, such as CycleGAN [67], BDL [100] and CyCADA [55]. We also qualitatively show keypoint detection of other animals where no labeled real images are available, such as elephants, sheep and dogs. Second, in or-

	Horse Accuracy							
	Eye	Chin	Shoulder	Hip	Elbow	Knee	Hoove	Mean
synthetic + real								
Real	79.04	89.71	**71.38**	91.78	82.85	80.80	72.76	78.98
CC-SSL-R	**89.39**	**92.01**	69.05	**92.28**	**86.39**	**83.72**	**76.89**	**82.43**
synthetic only								
Syn	46.08	53.86	20.46	32.53	20.20	24.20	17.45	25.33
CycleGAN [67]	70.73	84.46	56.97	69.30	52.94	49.91	35.95	51.86
BDL [100]	74.37	86.53	64.43	75.65	63.04	60.18	51.96	62.33
CyCADA [55]	67.57	84.77	56.92	76.75	55.47	48.72	43.08	55.57
CC-SSL	**84.60**	**90.26**	**69.69**	**85.89**	**68.58**	**68.73**	**61.33**	**70.77**
	Tiger Accuracy							
	Eye	Chin	Shoulder	Hip	Elbow	Knee	Hoove	Mean
synthetic + real								
Real	**96.77**	93.68	65.90	94.99	67.64	80.25	81.72	81.99
CC-SSL-R	95.72	**96.32**	**74.41**	**91.64**	**71.25**	**82.37**	**82.73**	**84.00**
synthetic only								
Syn	23.45	27.88	14.26	52.99	17.32	16.27	19.29	21.17
CycleGAN [67]	71.80	62.49	29.77	61.22	36.16	37.48	40.59	46.47
BDL [100]	77.46	65.28	36.23	62.33	35.81	45.95	54.39	52.26
CyCADA [55]	75.17	69.64	35.04	65.41	38.40	42.89	48.90	51.48
CC-SSL	**96.75**	**90.46**	**44.84**	**77.61**	**55.82**	**42.85**	**64.55**	**64.14**

Table 4.1: Horse and tiger 2D pose estimation accuracy PCK@0.05. Synthetic data are with randomized background and textures. Synthetic only shows results when no real image label is available, Synthetic + Real are cases when real image labels are available. In both scenarios, our proposed CC-SSL based methods achieve the best performance.

der to show the domain generalization ability, we annotated the keypoints of animals from Visual Domain Adaptation Challenge dataset (VisDA2019). In Section 4.4.3, we evaluate our models on these images from different visual domains. Third, the rich ground truth in synthetic data enables us to do more tasks beyond 2D pose estimation, so we also visualize part segmentation on horses and tigers and demonstrate the effectiveness of multi-task learning in Section 4.4.4.

Figure 4.3: Visualization of horse and tiger 2D pose estimation and part segmentation prediction. The 2D pose estimations are predicted using CC-SSL as described in Section 4.4.2 and part segmentation predictions are generated using the multi-task learning as described in Section 4.4.4. Best viewed in color.

4.4.1 Experiment Setup

Network Architecture. We use Stacked Hourglass [68] as our backbone for all experiments. Architecture design is not our main focus and we strictly follow parameters from the original paper. Each model is trained with RMSProp for 100 epochs. The learning rate starts with $2.5e^{-4}$ and decays twice at 60 and 90 epoches respectively. Input images are cropped with the size of 256×256 and augmented with scaling, rotation, flipping and color perturbation.

Synthetic Datasets. We explain the details of our data generation parameters as follows. The virtual camera has a resolution of 640×480 and field of view of 90. We randomize synthetic animal textures and backgrounds using

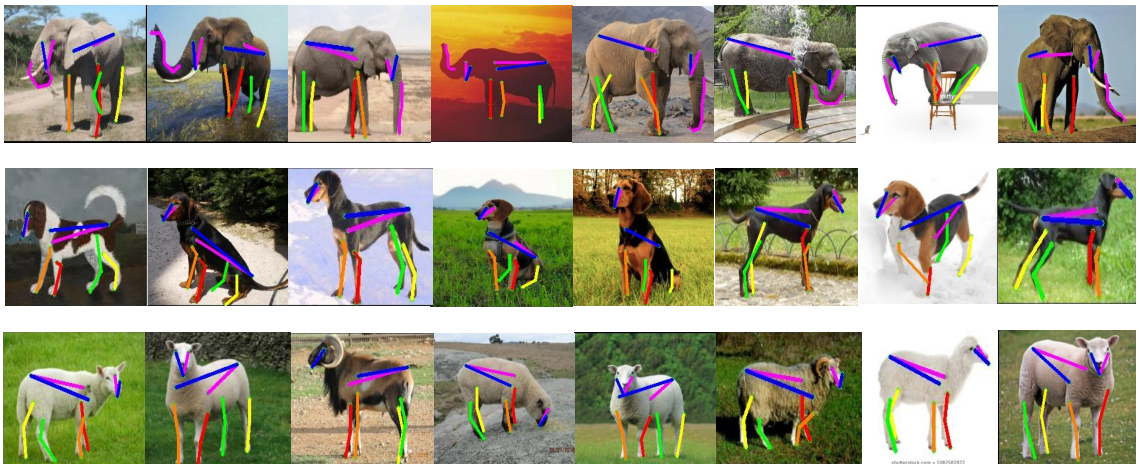

Figure 4.4: Visualization of 2D pose estimation of other animals. Our method can be easily generalized to elephants' trunks. Best viewed in color.

Coco val2017 dataset. We does not use any segmentation annotation from coco val2017. For each animal, we generated 5,000 images with random texture and 5,000 images with the texture coming with the CAD model, to which we refer as the original texture. We split the training set and validation set with a ratio of 4:1, resulting in 8,000 images for training and 2,000 for validation. We also generate rich ground truth including part segmentation, depth maps and dense 2D and 3D keypoints. For part segmentation, we define nine parts for each animal, which are eyes, head, ears, torso, left-front leg, left-back leg, right-front leg, right-back leg and tail. The parts definition follows [81] with a minor difference that we distinguish front from back legs. CAD models used in this chapter are purchased from UE4 marketplace[1].

CC-SSL In our experiments, we pick scaling and rotation from T_α and ob-

[1] https://www.unrealengine.com/marketplace/en-US/product/animal-pack-ultra-01

tain Δ using optical flow. λ_{decay} is set to 0.9 and we train one model for 10 epochs and re-generate pseudo labels with the new model. Models are trained for 60 epochs with γ set to be 10.0.

TigDog Dataset The TigDog dataset is a large dataset containing 79 videos for horses and 96 videos for tigers. In total, for horse, we have 8380 frames for training and 1772 frames for testing. For tigers, we have 6523 frames for training and 1765 frames for testing. Each frame is provided with 19 keypoint annotations, which are defined as eyes(2), chin(1), shoulders(2), legs(12), hip(1) and neck(1). The neck keypoint is not clearly distinguished for left and right, so we leave it out in all experiments.

4.4.2 2D Pose Estimation

Results Analysis. Our main results are summarized in Table 4.1. We present our results in two different setups: the first one is under the unsupervised domain adaptation setting where real image annotations are not available; the second one is when labeled real images are available.

When annotations of real images are not available, our proposed **CC-SSL** surpasses other methods by a significant margin. The PCK@0.05 accuracy of horses reaches 70.77, which is close to models trained directly on real images. For tigers, the proposed method achieves 64.14. It is worth noticing that these results are achieved without accessing any real image annotation, which

	Horse					
	Visible Kpts Accuracy			Full Kpts Accuracy		
	Sketch	Painting	Clipart	Sketch	Painting	Clipart
Real	65.37	64.45	64.43	61.28	58.19	60.49
CC-SSL	72.29	73.71	**73.47**	**70.31**	**71.56**	**72.24**
CC-SSL-R	**73.25**	**74.56**	71.78	67.82	65.15	65.87
	Tiger					
	Visible Kpts Accuracy			Full Kpts Accuracy		
	Sketch	Painting	Clipart	Sketch	Painting	Clipart
Real	48.10	61.48	53.36	46.23	53.14	50.92
CC-SSL	53.34	55.78	59.34	52.64	48.42	54.66
CC-SSL-R	**54.94**	**68.12**	**63.47**	**53.43**	**58.66**	**59.29**

Table 4.2: Horse and tiger 2D pose estimation accuracy PCK@0.05 on VisDA2019. We present our results under two settings: Visible Kpts Accuracy only accounts for visible keypoints; Full Kpts Accuracy also includes self-occluded keypoints. Under all settings, our proposed methods achieve better performance than baseline Real.

demonstrated the effectiveness of our proposed method.

We also visualize the predicted keypoints in Figure 4.3. Even for some extreme poses, such as horse riding and lying on the ground, the method still generate accurate predictions. The observations for tigers are similar.

When annotations of real images are available, our proposed **CC-SSL-R** achieves 82.43 for horses and 84.00 for tigers, which are noticeably better than models trained on real images only. CC-SSL-R is achieved simply by further finetuning the CC-SSL models using real image labels.

In addition to horses and tigers, we apply the method to other animals as well. The method can be easily transferred to other animal categories and we qualitatively show keypoint prediction results for other animals, as shown in

Figure 4.4. Notice that our method can also detect trunks of elephants.

We empirically find the performance does not improve much with CycleGAN. We conjecture that one reason is that CycleGAN in general requires a large number of real images to work well. However, in our case, the diversity of real images is limited. Another reason is that animal shapes of transferred images are not maintained well. We also try different adversarial training strategies. Though BDL works quite well for semantic segmentation, we find the improvements on keypoints detection is small. CyCADA also suffers from the same problem as CycleGAN. In comparison, CC-SSL does not suffer from those problems and it can work well even with limited diversity of real data.

We use the same set of augmentations as in [68] for baselines **Real** and **Syn**. We use a different set of augmentations for other experiments, which we refer to as Strong Augmentation. In addition to what [68] used, Strong Augmentation also includes Affine Transform, Gaussian Noise and Gaussian Blurring.

4.4.3 Generalization Test on VisDA2019

In this section, we test model generalization on images from Visual Domain Adaptation Challenge dataset (VisDA2019). The dataset contains six domains: real, sketch, clipart, painting, infograph, and quickdraw. We pick up sketch, painting and clipart for our experiments since inforgraph and quickdraw are

not suitable for 2D pose estimation. For each of these three domains, we manually annotate images for horses and tigers. Evaluation results are summarized in Table 4.2. Same as before, we use **Real** as our baseline. **CC-SSL** and **CC-SSL-R** are used for comparison.

For both animals, we observe that models trained using synthetic data achieve best performance in all settings. We present our results under two settings. Visible Keypoints Accuracy only accounts for keypoints that are directly visible whereas Full Keypoints Accuracy shows results with self-occluded keypoints.

Under all settings, CC-SSL-R is better than Real. More interestingly, notice that even without using real image labels, our CC-SSL method yields better performance than Real in almost all domains. The only one exception is the painting domain of tigers. We hypothesize that this is because texture information (yellow and black stripes) in paintings is still well preserved so models trained on real images can still "generalize". For sketches and cliparts, appearances are more different from real images and models trained on synthetic data show better results.

4.4.4 Part Segmentation

Since the synthetic animal dataset is generated with rich ground truth, our task is not limited to 2D pose estimation. We also experiment with part seg-

Models	Horse	Tiger
Baseline	60.84	50.26
+Part segmentation	**62.25**	**51.69**

Table 4.3: Horse and tiger 2D pose estimation PCK@0.05 with multi-task learning. We show models can generalize better to real images trained jointly using 2D keypoints and part segmentation.

mentation in a multi-task learning setting. All models are trained on synthetic images with Strong Augmentation and tested on TigDog dataset directly.

As shown in Table 4.3, we observe that models, trained on keypoints and part segmentation jointly, can generalize better on real images for both animals, compared to the baseline where models are only trained with keypoints. Since we cannot quantitatively evaluate part segmentation predictions, we visualize the part segmentation results on TigDog dataset as shown in Figure 4.3.

In the multi-task learning setting, we only make minor changes to the original Stacked Hourglass architecture, where we add a branch parallel to the original keypoint prediction one for part segmentation.

4.5 Conclusion

In this chapter, we present a simple yet efficient method using synthetic images to parse animals. To bridge the domain gap, we present a novel consistency-constrained semi-supervised learning (CC-SSL) method, which leverages both

spatial and temporal constraints. We demonstrate the effectiveness of the proposed method on horses and tigers in the TigDog Dataset. Without any real image label, our model can detect keypoints reliably on real images. When using real image labels, we show that models trained jointly on synthetic and real images achieve better results compared to models trained only on real images. We further demonstrate that the models trained using synthetic data achieve better generalization performance across different domains in the Visual Domain Adaptation Challenge dataset. We build a synthetic dataset contains 10+ animals with diverse poses and rich ground truth and show that multi-task learning is effective.

Part III

Model Diagnosis

Chapter 5

Controlling Hazardous Factors to Analyze Stereo Vision

In this chapter, we use synthetic data to diagnose the robustness of stereo algorithms. Despite the great progress in terms of average performance, stereo algorithms still fail easily under hazardous conditions, e.g. reflection, transparency. We utilize the controllability of object material in synthetic data to study model robustness to these hazardous factors.

5.1 Introduction

Stereo algorithms benefit enormously from benchmarks [107] [108]. They provide quantitative evaluation to encourage competition and track progress.

CHAPTER 5. CONTROLLING HAZARDOUS FACTORS TO ANALYZE STEREO VISION

Despite great progress over the past years, many challenges still remain unsolved, such as transparency, specularity, lack of texture and thin objects. These image regions are called hazardous regions [109] because they are likely to cause the failure of an algorithm. These regions are small and uncommon, but critical in the real world. For example, a street light is a thin object and covers a small region in an image, but missing it could be a disaster for autonomous driving.

Images in the real world contain different degrees of hazardous factors. A few images of KITTI dataset [110] contain reflective windshield or dark tunnel, making them more challenging than the others. To study algorithm performance on extreme conditions, more images can be captured [111] [25] under extreme weather conditions. But real images only contain sparse samples of different hazardous degrees. Although it would be possible to construct a huge image dataset which captures large degrees of all the hazardous conditions, the size of this dataset would be very large so that labeling all the hazardous regions of these images would be prohibitively expensive.

To address the problem of stress testing stereo algorithms, we developed a data generation tool which researchers can use to precisely control *nuisance factors* of a virtual scene, such as material properties, and produce new images. For example, in Fig. 5.1, we use this tool to vary the degree of specularity and show how this impacts the performance of a state-of-art stereo al-

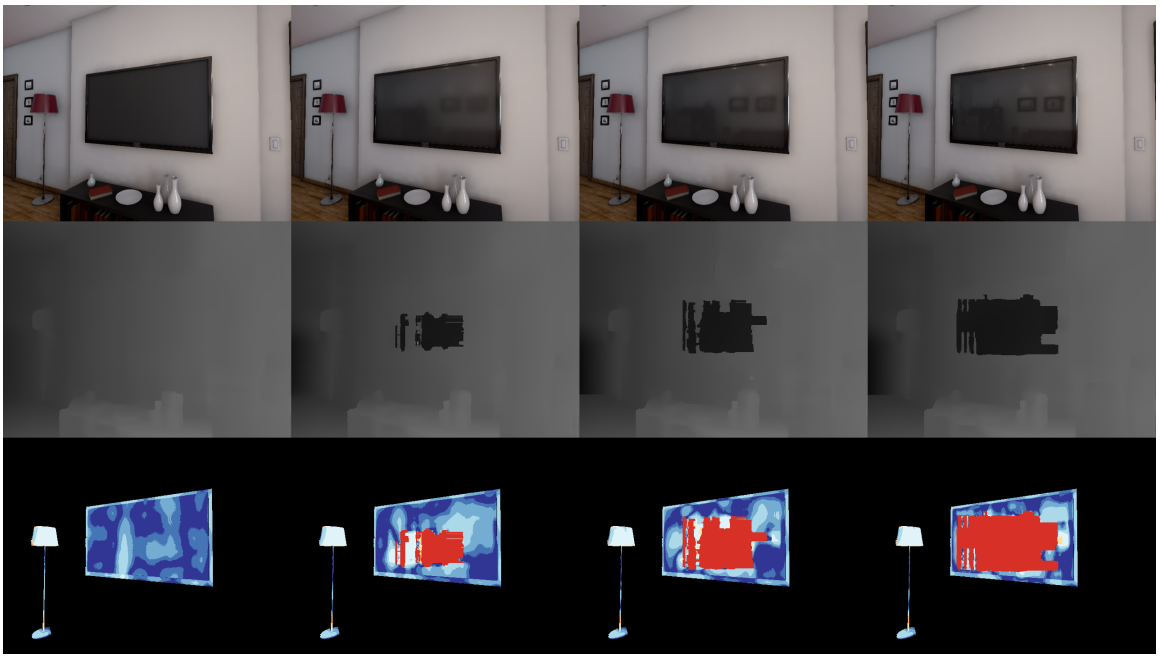

Figure 5.1: Different levels of specularity of the TV, from top to bottom are input image, disparity estimation and error compared with ground truth, the error is only computed for the specular regions. The visual difference in the first row is subtle, but is a very big challenge for state-of-art methods [2]. Best seen in color.

gorithm [2]. More generally, our approach enables us to follow the standard strategy of scientific research which changes variables separately and systematically and study their impact. In particular, we use this technique in our paper to study the relationship between hazardous factors and algorithm performance, which can help determine when an algorithm will break. These findings from synthetic images can then be validated using real images, but this validation requires a smaller amount of testing images (hence avoiding the need for excessive annotation of real images). Using virtual worlds as a virtual laboratory is conceptually simple, but practically very hard. It requires constructing realistic 3D content and easy-to-use tools. These challenges are addressed in this chapter. Our experiments focused on stereo algorithms, but this idea can be applied to other computer vision algorithms.

In this chapter, we use this synthetic image generation tool to study the effect of four important hazardous factors on stereo algorithms. These hazardous factors are chosen to violate some of the basic assumptions of traditional stereo algorithms. For example, specular and transparent surfaces violates the brightness consistency constraint, which assume that the intensity properties of corresponding points are similar (because specularity means that the intensity of a surface point will depend on the viewpoint). Although these hazardous factors are well-known to the community, there have been few attempts at quantitative evaluation of the impact of individual factor due to challenges

of annotating these factors. We validate the findings on the synthetic images by comparing with a real world dataset KITTI, which we also annotated for this purpose. We were inspired by the theoretical framework to analyze hazardous factors proposed in [109], but their framework required a lot of manual annotation of hazardous regions of images which we reduce. In short, our tool can produce binary mask of hazardous regions automatically, making their theoretical framework practical.

For our approach to succeed it is necessary for our synthetic data to be realistic. This is possible firstly because the 3D scenes in advanced synthetic datasets are increasingly naturalistic. The 3D scenes we use are produced by the virtual reality industry and are constructed to mimic real world configurations as closely as possible. Secondly, the rendering quality is also very high. The physical based materials and precomputed lighting of Unreal Engine prevent the sharp shadows and plain material observed in many low quality synthetic images. The difference of realism is subtle but very important for our purpose, see Fig. 5.1.

To summarize, in this chapter we developed a data generation tool called UnrealStereo and used it to stress test stereo algorithms. The main contributions of our paper are as follows: Firstly, we provide a tool to enable researchers to control the nuisance factors in a virtual environment to analyze stereo algorithms. Secondly, hazardous regions are automatically produced in

our framework, making the theoretical framework in [109] practical. Third, we controlled the hazardous factors to show the characteristics of different stereo methods and validated our result on novel annotations of the KITTI dataset. Our tools are open source and will be made available to the community.

5.2 Related Work

Many stereo datasets have been created for training and evaluating stereo algorithms. The Middlebury stereo dataset [107] is a widely used indoor scene dataset, which provides high-resolution stereo pairs with dense disparity ground truth. The KITTI stereo dataset [112] [110] is a benchmark consisting of urban video sequences while semi-dense disparity ground truth along with semantic labels are available. Due to demand of complex equipment and expensive human labor, these two real-world datasets have relatively small sizes. The larger KITTI dataset has about 400 labeled stereo pairs in total for public use. Besides providing much more image pairs and ground truth for training and evaluation. We provide the ability to control nuisance factors in an image. Also our generated hazardous regions are useful for analysis.

Synthetic data has attracted a lot of attention recently. The progress of computer graphics makes synthesizing realistic images much easier. The ability to get a large amount of images and ground truth is attractive. Synthetic

data have been used in optical flow [24], semantic segmentation [16] [25] [47], stereo [113]. Images and ground truth are provided in these datasets, but the virtual scenes are not available for various reasons. So it is not possible to render new images or change the properties of these scenes. Instead of constructing virtual scenes from scratch, we use game projects that are publicly available in the marketplace. Our tool enables generating images and ground truth from these game projects. Anyone can use our tool to tweak these virtual scenes, e.g., by varying the hazardous factors, and hence produce more data. Many virtual scenes constructed by visual artists in the marketplace can be used. Different from rendering images from a commercial game binary [47], the ability to access 3D models enables us to modify the scene, generate more ground truth and do various virtual experiments. Unlike Sintel [24] and Flyingthing3D [113], our approach utilizes more realistic 3D models and designed for real-time rendering. Previously, OVVV [114] attempted to use synthetic data for evaluating algorithms, but there is no easy-to-use platform available. UnrealCV [27] is an open source tool to generate synthetic images. It has been used to diagnose detection algorithms.

Understanding the robustness of stereo algorithms is important. In the HCI/Bosch robustness challenge [111], challenging images are captured. In order to test algorithm in different conditions in a controlled way, lab setup based on toys and robotics arm is created [115] to control nuisance factors, but the im-

ages are very different from normal conditions. CV-HAZOP [109] proposes the idea of analyzing hazardous factors in an image. Their method requires manually annotating risk factors, such as specularity area, from images, which is difficult to perform and hard to scale up. Our synthetic pipeline can automatically identify these hazardous regions, enables large-scale analysis. The ability to control the severity of hazardous factors also helps us to better understand the weakness of an algorithm.

5.3 Hazardous Factor Analysis

In this section, we first described the data generation tool UnrealStereo. Then we varied the nuisance factors to produce hazardous regions to stress test state of the art stereo algorithms. Finally, hazardous regions are computed for images rendered from realistic 3D scenes to analyze the impact of each hazardous factor.

5.3.1 UnrealStereo Data Generation Tool

Our data generation tool is based on Unreal Engine and UnrealCV [27]. Unreal Engine is an open-source game engine, which provides 3D design tools for game developers. UnrealCV is an open source tool to extract information from an Unreal Engine project.

UnrealCV showed the idea of using realistic video game content for computer vision research, the concept is proved with a small indoor scene. Applying UnrealCV to the stereo task is non-trivial and requires a lot of engineering. 1. The support of multiple cameras is implemented. The second camera automatically follows the first camera and keeps relative position fixed. The distance between two cameras can be adjusted to simulate different configurations. 2. UnrealStereo supports nuisance factor control, such as material. This requires a deep understanding of the material system of Unreal Engine. 3. The depth of transparent objects is stored in a different way in Unreal Engine. We fixed this issue to get accurate depth for transparent objects. 4. Special visual effects such as the animation and particle effects are correctly handled. 5. Many scenes are tested to ensure compatibility. 6. Scripts are developed to support the generation of a large quantity of images. The development took us a few months and will be released to the community.

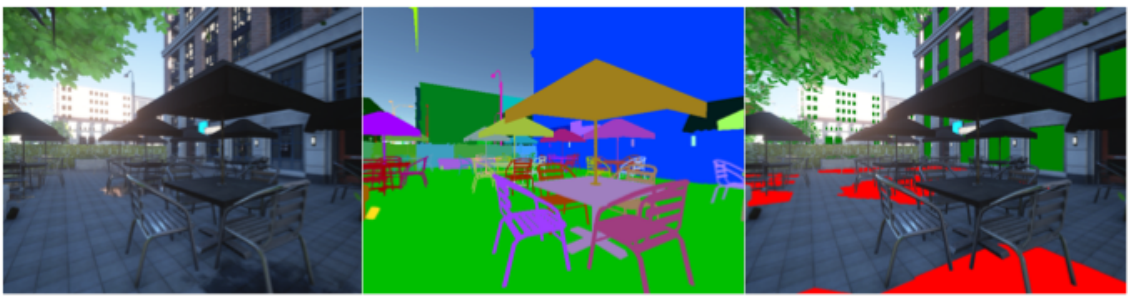

Figure 5.2: From left to right are rendered images, object instance mask, material information (green shows transparent and red shows specular region).

The image and depth are captured from the 3D scenes for both two cameras,

besides other extra information shown in Fig. 5.2. Given a rectified image pair, the goal of stereo matching is to compute the disparity d for each pixel in the reference image. The disparity is defined as the difference in horizontal location of a point in the left image and its corresponding one in the right. Then the conversion between depth z and disparity d is shown in the following relation $z = \frac{fB}{d}$. where f is the focal length of the camera and B is the baseline that is the distance between the camera centers. The correctness of disparity is verified by warping the reference image according to its disparity map and comparing it with the target image.

None of previous synthetic datasets for stereo [47] [24] [25] [113] allow users to generate more images, because the 3D scenes and tools for rendering are not provided. The ability to generate images from a virtual scene is more useful than a pregenerated image dataset. Users can render more images about a hazardous case they care about. Even with the access to 3D scenes, controlling the nuisance factors of a 3D scene still requires professional knowledge. Our tool can not only allow users to generate more images, but also control nuisance factors of images. Using a virtual world to do experimental control is conceptually simple, but practically hard. It requires the 3D scene files and the knowledge to use professional 3D modeling tools, such as blender. This challenge is addressed in our tool.

Our data tool is designed to be compatible with any Unreal Engine project.

This enables us to use many realistic 3D scenes from the game and VR industry to generate images. These images are realistic in two ways. First, they contain correct semantic information. The 3D scenes we use are created to mimic the real images as close as possible. Second, the rendering engine can produce very good indirect lighting, so that the shadow and reflection are very realistic.

5.3.2 Designing Hazardous Cases for Evaluation

Most of stereo algorithms can be formulated in terms of minimizing an objective function w.r.t disparity d,

$$E(d) = \sum_{\boldsymbol{p}} E_d(d(\boldsymbol{p})) + \lambda \sum_{(\boldsymbol{p},\boldsymbol{q})\in\mathcal{C}} E_s(d(\boldsymbol{p}), d(\boldsymbol{q})) \tag{5.1}$$

where the data term E_d usually represents a matching cost and the smoothness term E_s encodes context information within a support region $\mathcal{C}$ of pixel p (q is a pixel in $\mathcal{C}$). Local stereo methods [116] [117] do not have a smoothness term and utilize only local matching cues. Global methods [118] [119] [120] [2] impose smoothness constrains on neighboring pixels or superpixels by adding a smoothness term.

The success of these methods relies on some basic assumptions hold for the scene they encounter. First, the single image layer assumption is required by most methods to do correspondence between binocular image pairs. Second,

the local surface should be well-textured for matching algorithms to extract feature. Third, the smoothness term in global method functions under the assumption that the disparity vary slowly and smoothly in space. However, these assumptions can easily be broken in real world scenarios. For example, specular and transparent surfaces would create multiple image layers which breaks the first assumption. Textureless objects are everywhere such as white walls and objects under intense lighting. Besides, smoothness assumption does not hold for regions with many jumps in disparity, e.g. fences and bushes.

Since specularity, transparency, weak texture, frequent disparity discontinuities as well as occlusion often break the assumptions of most stereo methods, we call them hazardous factors following [109]. Special efforts have been made to resolve these difficulties in recent years. Nair *et al.* [121] derive a data term that explicitly models reflection. Güney *et al.* [122] leverage semantic informations and 3D CAD models to resolve stereo ambiguities caused by specularity and no texture. An end-to-end trained DCNN based algorithm [113] performs well on specular regions of KITTI stereo 2015 [110] after finetuning on the training set.

Evaluating stereo algorithms under different hazardous factors on real data is highly inconvenient, because publicly available datasets are limited both in quantity and variety of hazardous factors. Thus, we propose to use synthetic data as alternative. The UnrealStereo tool we developed is able to produce haz-

(a) Specularity (b) Lack of texture (c) Transparency (d) Disparity jumps

Figure 5.3: From (a) to (d) are cases we designed to test algorithms. They are specularity, lack of texture, transparency and disparity jump . In (a), the screen of a TV is set to be specular. In (b), the wall and the ceiling in the room are made textureless. In (c), the sliding wall has a transparent surface. In (d), objects such as bamboos, fences and plants give frequent disparity discontinuities.

ardous cases in the synthetic with lighting and material controlled, making it tractable to conduct precise evaluation. As a demonstration, we establish four virtual scenes with high reality each of which includes one factor. Stereo image pairs are rendered from various viewpoints together with dense disparity groundtruth. Fig. 5.3 shows the outlines of the four scenes.

In the specularity challenge(Fig. 5.3(a)), we obtain the specular effect by diminish the roughness of specific materials. The major specular object is the screen of a TV towards which test stereo pairs are obtained from various viewpoints and distances. In the lack of texture challenge(Fig. 5.3(b)), the wall and the ceiling in the room are made textureless because they are the most common textureless objects in real world. To achieve texturelessness while keep reality, we do not directly remove the material of the walls but increase the smoothness of their material. A set of ten test image pairs are collected covering different orientations to the surrounding textureless walls. In the trans-

parency case(Fig. 5.3(c)), we placed a transparent sliding wall in a room. In the disparity jumping case(Fig. 5.3(d)), we place in the scene objects such as bamboos, fences and plants of various sizes and poses, which easily form many disparity discontinuities distributed within a small region. Details and results for evaluation on these cases are presented in Section 5.4.1.

One of the factors available to control is the area of textureless regions. This is crucial to stereo methods because as the textureless region gets larger, it becomes more difficult for the smoothness term to use context information such as the disparity of neighboring well-textured objects. For specularity, transparency and lack of texture factor, our tool also allows investigation into how performance of algorithms degrades as the extent of challenge increases. Specifically, by controlling the opacity of an object, it is possible to vary the extent of transparent while keeping the rest of the scene intact.

Because synthetic and real data are in different domain, after receiving the evaluation results on virtual scenes, it is important to verify them on real-world dataset. To this end, for specularity and no texture factor, we manually annotated corresponding regions on KITTI 2015 [110].

5.3.3 Automatic Hazardous Region Discovery

It is important to pay more attention to hazardous regions of an image, because these regions are most likely to cause the failure of an algorithm. Manu-

ally designed hazardous cases are important for understanding an algorithm, but the size of manually designed data is not large enough.

On the other hand, due to the popularity of virtual reality, there are a lot of high quality virtual environments. These virtual environments can be purchased with a fair price (less than $50) or even free. These virtual environments can produce large amount of images. Moreover, we can also access other information, such as object instance mask, material property. We use these information to compute hazardous regions and to evaluate stereo algorithms on these hazardous regions.

Our rendering process produces extra information beyond depth information. These information includes: object instance mask, specular region, transparent region. Using these extra information, we can locate these hazardous regions automatically. Specifically, we add new focus to stereo method evaluation by providing binary masks for the these regions: 1. weakly textured material, 2. specular material, 3. transparent material. These regions are related to what we mention in Section 5.3.2 respectively. Fig. 5.4 shows an example of these masks. All of above masks are generated automatically from the object segmentation ground truth. For each object, we annotate its material information only once, before rendering process, then no more human effort is required to obtain corresponding masks.

We establish a large dataset using six publicly available game scenes. They

(a) (b) (c) (d)

Figure 5.4: Binary masks that we compute from object mask and material property. From (a) to (d) are: mask for non-occluded region, object boundary region, textureless region and specular region. Best seen in color.

are a small indoor room used in UnrealCV [27], a large temple scene, three houses and one block of street. There are different layouts in these houses such as living room, kitchen and bathroom. The largest scene contains more than 1,000 objects while hundreds on average, including those high reflectance such as mirrors, bathtubs and metal statues, transparent objects such as glass, glassdoors and windows. Subtle visual effects are included, e.g. leather sofa and the specular surface of a TV. Screenshots of these games can be seen in Fig. 5.6. Specifically, for each game scene we record a video sequence that covers different viewpoints in the environment, which results in 10,825 image pairs in total.

The comparison with other stereo datasets is shown in Table. 5.1. Besides depth and disparity ground truth, we provide extra information, such as: object segmentation mask and material properties. The unique feature of our dataset is the nuisance factors of virtual worlds can be controlled with our tool and more challenging images can be produced as shown in Fig. 5.5. Instead of

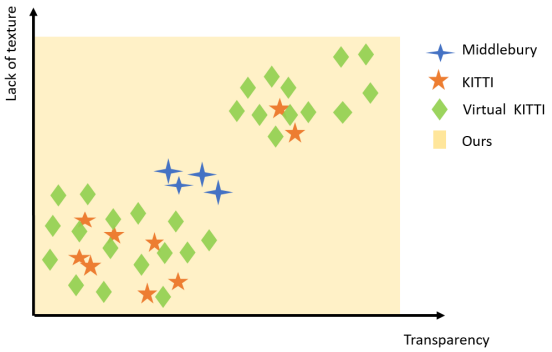

Figure 5.5: While other stereo datasets do provide images containing hazardous regions it is hard for them to densely vary the degrees of the hazardous factors. By contrast, UnrealStereo enables users to control hazardous factors to cover a much wider range of cases. Then real world datasets can be used to validate the findings on the synthetic datasets. Also findings on synthetic datasets may indicate particularly challenging degrees of the hazardous factors, which can be investigated by constructing real world datasets tuned to these precise degrees of hazard.

just providing an image dataset with fixed number of images, we provide a synthetic image generation tool. This tool can be used to design new hazardous cases, generate more images. More game scenes from the marketplace can be used in experiment.

5.4 Experiment

We choose five types of state-of-the-art stereo algorithms to evaluate on the challenging testing data we rendered. They are representatives of local methods ELAS [116] and local method with spatial cost aggregation CoR [2], global methods on pixel-level MC-CNN [120] and superpixel-level SPS-St [119] as

	Density	Scene Type	Extra Info.
Middlebury [107]	dense	laboratory	
KITTI2012 [112]	50%	outdoor	seg.
KITTI2015 [110]	50%	outdoor	seg.
Virtual KITTI [47]	dense	outdoor	seg.
FlyingThings3D [113]	dense	toy, outdoor	seg., material
Ours	dense	indoor, outdoor	seg., material

Table 5.1: Comparison of stereo benchmarks and datasets. Our data generation tool can provide rich information, such as: object segmentation mask, material properties. It can also be used to produce new challenging images from a virtual world

Figure 5.6: The six virtual scenes we use in our experiments, from left to right are image, depth and object mask. These virtual scenes are purchased from Unreal Engine marketplace.

well as end-to-end CNN based method DispNetC [113]. Implementation from the authors of these methods are adopted. For the weights of the MC-CNN, we adopt the models used in their submission to KITTI. While for DispNetC, the original model trained on the synthetic dataset FlyingThings3D [113] is adopted. In our experiments, we mainly adopt the end-point error (EPE), which takes average of the absolute difference from ground truth, as error metrics. In Section 5.4.2 we provide results in the 3 pixel error, i.e. the percentage of pixels which deviate from ground truth more than 3 pixel. The 3 pixel error metric is proposed to cover calibration and laser measurement errors in real-world datasets and we use it here to compare results with them.

5.4.1 Hazardous Cases Evaluation

We use 10 different viewpoint for each of the hazardous cases we designed, i.e. specular, semi-transparent, textureless, and disparity jumps, covering both fronto-parallel and slanted surfaces. At each viewpoint of hazardous scenes except disparity jumps case, we start from the easiest parameter settings that are roughest, opaque or well-textured and adjust the corresponding parameter step by step to increase the extent of hazard, resulting in 8 different levels of corresponding hazard per viewpoint and a total of 80 stereo image pairs. In specularity, transparent and textureless cases, only regions labeled as these types are evaluated. For all the cases, we evaluate non-occluded regions be-

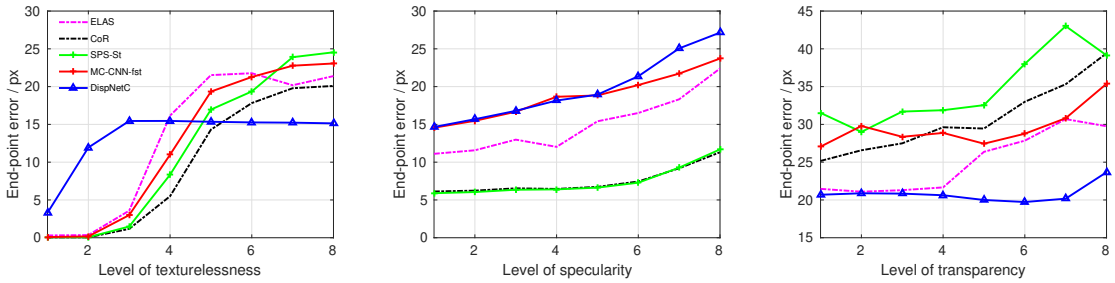

Figure 5.7: The influence of texturelessness, specularity and transparency at different levels. The performance of stereo algorithms is evaluated in terms of end-point error. Larger number of level represents is controlled by parameters for corresponding materials. Each data point represents an average over 10 different viewpoint.

cause occlusion will introduce new problems to the task while we focus on analyzing one factor at a time.

Fig. 5.7 displays the comparison of degradation of state-of-the-art stereo algorithms. Table 5.2 and 5.3 show quantitative results of their performance. To verify our result, we annotate specular and textureless regions on KITTI training set and compared the performance of selected methods with that on our data. To annotate hazardous regions of KITTI dataset, annotators are asked to mask specular and textureless regions images with photoshop selection tool. The ground truth for transparent regions on KITTI is incomplete, so we are not able to compare performance on other hazardous factors. Table 5.2 provides the quantitative results of this comparison.

As shown in Table 5.2, results from both our data and KITTI show that performance of all evaluated methods on hazardous regions drops dramatically

	Specular			Textureless			Overall
	High	Med	KITTI	High	Med	KITTI	KITTI
ELAS [116]	22.39	12.03	4.07	21.40	16.16	7.07	1.55
SPS-St [119]	11.69	**6.40**	2.71	24.54	8.34	2.13	1.23
CoR [2]	**11.32**	6.46	**2.61**	20.09	**5.50**	**1.98**	1.11
MC-CNN-fst [120]	23.72	18.66	2.62	23.06	10.99	3.14	**1.10**
DispNetC [113]	27.15	18.15	3.42	**15.14**	15.46	3.48	1.59

Table 5.2: Performance on hazardous regions on our data and KITTI training set in end-point error (EPE). Hazardous levels of medium (Med) and high are presented for our data. Only regions annotated as corresponding hazard are evaluated.

	Transparent	Disparity Jumps
ELAS [116]	29.76	8.43
SPS-St [119]	39.14	9.06
CoR [2]	39.41	8.62
MC-CNN-fst [120]	35.37	8.57
DispNetC [113]	**23.67**	**6.63**

Table 5.3: Performance for extreme transparent and disparity jumps hazard in end-point error. Errors in disparity jump case are in term of the full image.

compared with overall results, indicating that these methods are not robust enough. Moreover, such results on KITTI also suggest that hazardous cases are uncommon in real world, but easy to cause failures. Errors on our data is higher, because hazardous regions we design are usually larger than those on KITTI, posing bigger challenge to the regularization techniques. It is also worth noticing that methods that perform better on our data also do well on KITTI. For textureless factor in particular the performance ranking of medium level matches that on KITTI. This suggests that results on our data generalize well to the real world.

Looking into Fig. 5.7 and the quantitative results, we discover more information about these methods when facing each hazardous factor.

Specularity: CoR outperforms other methods for specularity factor, closely followed by SPS-St. As shown in Fig. 5.7, they exhibit high robustness as the material becomes more specular, and CoR achieves the lowest error on corresponding regions on KITTI. This lead to a conclusion that cost aggregation on suitable regions or regularization on superpixels can to some extent reduce the vulnerability to matching ambiguities. MC-CNN uses CNN to compute matching cost and works well on normal cases, but suffers easily on specular surfaces.

No texture: Large support regions also helps regularize the result on textureless regions, so CoR and SPS-St work well. The robustness of DispNetC

on extremely textureless regions is expected. Because they have large receptive fields to incorporate context information. As the texture of objects become more evident, they become less competitive which is reflected on KITTI results. Purely local method fails easily on these regions as the consequence of missing or ambiguous local cues. Moreover, pixel level smoothness terms that MC-CNN utilize are not enough to capture large context, so they fail too.

Transparency: Errors on transparent regions of synthetic data are the highest as can be observed. This poor performance on transparent regions is not confusing as none of these methods make efforts to handle multiple image layers. It is conceivable that DispNetC method fails because it has not been trained with such kind of data.

Disparity jumps: DispNetC outperforms others because it does not explicitly impose smoothness constraints, which helps to avoid erroneous over-smooth. CoR and SPS-St perform worst on disparity jumps. This suggests that larger support regions introduce errors on depth discontinuities when they are not adaptive enough.

5.4.2 Automatic Hazardous Region Discovery

We evaluate these algorithms on a testing set including 484 stereo image pairs which are randomly sampled from the 10k images from the six virtual scenes. The average performance on full image, non-occluded regions and

	Full		Non-Occluded		Specular		Textureless		Transparent	
	EPE	>3px	EPE	>3px	EPE	>3px	EPE	>3px	EPE	>3px
ELAS [116]	12.08	30.7%	9.09	24.5%	8.18	22.5%	14.04	53.5%	11.48	41.5%
SPS-St [119]	7.93	22.0%	5.16	15.3%	**6.46**	**16.2%**	10.74	40.6%	**9.94**	**35.9%**
CoR [2]	7.74	22.8%	4.97	15.9%	7.07	24.3%	8.33	42.1%	10.07	36.2%
MC-CNN-fst [120]	**7.64**	**20.6%**	**4.62**	**13.2%**	6.94	16.9%	7.62	37.6%	10.52	36.0%
DispNetC [113]	7.98	31.2%	5.96	24.9%	7.84	29.0%	**6.02**	**33.8%**	12.75	51.3%

Table 5.4: Performance of state-of-the-art stereo algorithms on test set of rendered dataset. Errors in full image, non-occluded, specular, textureless and transparent regions are included. Both end-point error (EPE) and ≥ 3 pixel error are evaluated by applying the masks proposed in Section 5.3.3.

three types of hazardous regions, i.e. specular, textureless and transparent, are shown in Table. 5.4. We also compare the results with overall performance on KITTI in Table. 5.2.

For overall performance in full and non-occluded regions in EPE, the best three methods and their ranking on this testing set are identical to those on KITTI. The overall errors are higher on our data. There are two possible causes. One is that the percentage of hazardous regions on our dataset is significantly larger than KITTI. The other is that KITTI only provides semi-dense ground truth, which excludes many hazardous regions, i.e. the windows of cars.

For all method, errors on hazardous regions are consistently larger than non-occluded regions, except for ELAS tested on specular regions. This exception is explainable because large errors on other hazards can elevate the error on non-occluded region which only excludes the influence of occlusion.

Local method ELAS receives poor performance on all hazardous regions,

which is not surprising because neither strong regularization techniques nor special modeling is included in it. The analysis in Section 5.4.1 suggests that non-local regularization using large support regions would reduce the influence of matching ambiguity. Here the top performance of SPS-St and CoR on specular and transparent regions verify that point. That DispNetC outperforms others on textureless region could result from the level of texturelessness, since Fig. 5.7 shows that DispNetC is robust on extremely textureless scene. As the state-of-the-art on KITTI, MC-CNN also achieves top performance on full and non-occluded regions. But its performance on hazardous regions is mediocre.

The relation of the two error metrics is worth noticing. End-point error (EPE) measures the error in average, while 3 pixel error measures the percentage of incorrect pixels. They are not in agreement in some cases, which could reveal characteristics of some algorithms. Specifically, for DispNetC, significantly higher 3 pixel errors is observed. The L1 loss function used in its training process encourages lower EPE, which could account for the discrepancy.

5.5 Conclusion

In this chapter, we presented a data generation tool UnrealStereo to generate synthetic images to create a stereo benchmark. We used this tool to an-

alyze the effect of four hazardous factors on state-of-the-art algorithms. The hazardous factors were specularity, lack of texture, disparity jumps, and transparency. Each factor was precisely controlled to see its impact. They were varied at different degrees and even to an extreme level to produce corner cases. We also tested several state-of-the-art algorithms on six realistic virtual scenes. The hazardous regions of each image were automatically labeled from the ground truth, e.g., the object mask and the material properties. We found that the state-of-the-art method MC-CNN [120] outperforms others in general, but lacks robustness in hazardous cases especially specular regions. DCNN based method [113] exhibits interesting properties due to its awareness of larger context. We also validated our findings by comparison to results on the real world KITTI dataset where we manually annotated the hazardous regions. We note that an advantage of synthetic data generations tools is that they enable us to explore many degrees of hazardous factors in a controlled setting, so that the time-consuming manual annotation of real images can be reduced. Perhaps manual annotation will only be needed in a limited (sparse) number of cases in order to validate the results on synthetic images. Results on synthetic datasets may be able to isolate those degrees of hazardous factors which particularly hurt the performance of algorithms, enabling us to only do manual annotation of real images in these particular cases. We note that identifying the failure modes of stereo algorithms, in particular to hazardous

factors, can be used to help develop better algorithms by, for example, providing additional training data.

Our data generation tool can be used to produce more challenging images and is compatible with publicly available high-quality 3D game models. This makes our tool capable for many applications other than stereo. In our future work, we will extend our platform to include more hazardous factors such as the ratio of occlusion and analyze more computer vision problems. It is also interesting to explore the rich ground truth we generate, such as object mask and material properties. This semantic information will enable the development of computer vision algorithms that utilizes high-level knowledge, for example like the recent stereo algorithms that use 3D car models.

Chapter 6

Understand What Activity Classification Models Have Learnt

This chapter applies synthetic data to diagnose an activity classification model. The diagnosis focuses on whether the model uses essential factors (human motion) or nuisance factors (background, context object) to perform the classification. This chapter extends the idea in Chapter 5 and present a generic framework called identity preserved transform (IPT). The sensitivity to IPT can indicate the robustness of a model in real-world applications.

6.1 Introduction

We have witnessed the rapid development of video activity classification models thanks to the thriving of Convolution Neural Networks in recent years [123] [124] [125]. However, a video activity classification model trained on one dataset often failed to generalize to another [126]. Activity classification model can solve the task easily by fitting to correlated factors. For example, the model can associate the activity with typical backgrounds where the activity happen or clothes style, rather than trying to parse the human pose. This poses a challenge for the vision model, which requires the model to successfully extract abstract information, rather than using low-level features.

Human activity, in most cases, can be defined by the human motion instead of **nuisance factors**, such as the human appearance and the environment. Capturing the **essential factors** (human motion) is vital for a robust activity classification model. Psycho-physical experiments show that human can reliably recognize the activity by watching moving dots. We expect a well-performing activity classification model to have similar properties. Researchers created larger datasets [124] [126] and carefully selected the combination of nuisance factors [115] [127] to study the model robustness. In addition to these efforts, we propose a framework to understand a model.

We propose a method called **Identity Preserve Transform (IPT)** to in-

spect what an activity classification model has learnt. IPT is a transform which manipulates nuisance factors of an image, while keeping the essential factors the same. This type of transform is inspired by the image generation procedure. It includes two types: image-space transform and 3D transform. Image-space transform is applied on a generated image, while 3D transform affects an image by directly changing the underlying nuisance factors. We use both image-processing techniques and a computer vision model providing prior knowledge (semantic transform) for image-space transform. In order to achieve 3D transform, we implemented a synthetic data generation pipeline,which takes a combination of rendering parameters to render a realistic virtual human and nuisance factors can be directly manipulated.

Identity Preserve Transform unveils interesting properties of state-of-the-art models. Take Temporal Segment Network (TSN) as an example. The model is sensitive to small perturbation created by image-processing. Note that we did not specifically target the model with adversarial attack [6]. More interestingly, the quantitative result shows the model makes decision mainly on the object or background of the video, rather than using the human pose. This can be further validated with visualization technique [3]. This explains why the high performance on the trained dataset does not generalize to other datasets and real-world scenarios. Our powerful synthetic data pipeline enables us to further analyze the relationship between the model performance and certain

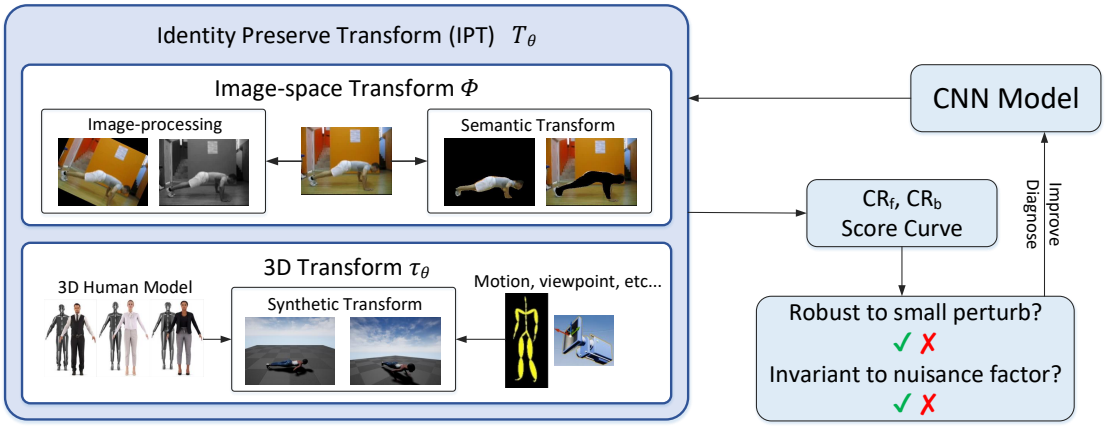

Figure 6.1: This figure illustrates an overview of Identity Preserve Transform (IPT). IPT consists of two types. It can take an activity classification model as input and be used to analyze properties of the model. The analysis can be used to improve model design.

factors. The observation for synthetic data can be easily verified using a small real dataset. The IPT operates only on the input data regardless of the model architecture, so it can easily be adopted to study other models.

Our contribution can be summarized as follows: 1. We propose identity preserve transform, a method to inspect an activity model using data probes and independent of model architecture. 2. We analyze a state-of-art model and showed it does not classify video activities according to human motion. 3. We collect a synthetic activity video dataset and develop a diagnostic toolkit to perform IPT. The source code will be available to help others understand and develop activity classification models.

6.2 Related Work

There are both qualitative and quantitative methods for understanding deep CNNs-based models, the qualitative type mainly focuses on visualizing intermediate layer feature maps and visual saliency, while the quantitative type explains the model through feature importance scores or factor importance scores.

Methods based on feature importance scores alter individual features (pixels, super-pixels, word-vectors, etc) through removal or perturbation [128] for each input to the model to approximate the importance of each feature for model's prediction. They have been proved susceptible to human confirmation biases [129]. Whereas Aubry *et al.* [130] analyzed CNN feature responses corresponding to different scene factors (object color, object style, lighting, etc.) by controlling them via rendering using a large database of 3D CAD models. In our approach, we abstract influential elements from the generation of human activity videos instead of selecting scene factors.

Recently, researchers started to use synthetic data (data generated through computer graphics) to understand vision models. This is mainly due to the high cost and difficulty to collect and annotate large numbers of controlled real data. Synthetic data has been used to study the sensitivity to rendering parameters, such as viewpoint [131] [132], material property [22]. The controllability of the

synthetic data also enables studying the invariance and equivariance property of a model [130].

6.3 Method

6.3.1 Identity Preserve Transform

Our design of Identity Preserve Transform is inspired by the data generation process. There are various factors influencing what the data looks like. Theoretically, all factors can be controlled during the data generation process.

Given a computer vision task, the entire set of factors can be split into 2 subsets. One is those factors directly related to the task which represent the most essential information for the task, such as the human motion for human activity classification in our case, or object shape and texture for object detection. We denote this subset as p. The other subset comprises of nuisance factors not directly related to the task, and a model should be insensitive to their change or even to their absence, such as viewpoint, human appearance for human activity classification and background color for object detection. We denote the latter subset as θ.

Therefore, the data generation process can be modelled by Eq. 6.1, where G denotes generation function, I denotes the data generated.

$$I = G(p, \theta) \tag{6.1}$$

If we manipulate p with τ_p and θ with τ_θ during data generation process, effects on the generated data I is equivalent to a transform T after the generation, as in Eq. 6.2.

$$T(I) = G(\tau_p(p), \tau_\theta(\theta)) \tag{6.2}$$

A well performing model capable of learning the essential information for a specific class (denoted by p) should be invariant or insensitive to changes of the nuisance factors (denoted by θ). It means that as long as p keeps constant, no matter how τ_θ changes θ, a good model should yield same results, as in Eq. 6.3. Here we denote the model as f.

$$f(I) = f(T_\theta(I)) = f(G(p, \tau_\theta(\theta))) \tag{6.3}$$

Identity Preserve Transform (IPT) refers to transformations that preserves p. It can be implemented in the image space or in the 3D space. Hence, IPTs can be categorized into two types: one is image-space transforms, which operate on the generated data I, the other is 3D transforms, which operate on θ during data generation process.

The identity factor p for video activity classification is human motion. It is

the most essential information a model should learn from training data. Compared with human motion, background, human appearance, viewpoint and lighting, etc. are all nuisance factors for classifying video human activities, they belong to θ. If a human activity classification model can develop a mechanism to model and infer human motion, it should be insensitive to the change in all these nuisance factors.

6.3.2 Image-space Transform

Image-space transform is IPT operating on generated data to simulate the change of nuisance factors during data generation process, as in Eq. 6.4 Image-space means it edits image extracted from a video.

It can be realized by applying image processing techniques commonly used for data augmentation, such as adding noise, blurring, we call it image processing transform. It can also be realized by a computer vision model providing prior semantic knowledge, such as segmenting objects in the image, super resolution, etc., called semantic transform.

$$T_\theta(I) = \Phi * G(p, \theta) \tag{6.4}$$

6.3.2.1 Image Processing Transform

Image processing transform influences how a video look like while keeping background and human motion the same. For example, we can lower the image resolution by applying a blurring filter, or increase the image brightness through histogram equalization. However, lowering the image resolution or increasing brightness would not change people's body motion inside the video.

If a video activity classification model is robust enough, we expect its performance not to be harmed by image processing transform on the test data. Once its classifying accuracy drops due to image processing transforms, we should be alarmed that it overfits to image pattern without really learning human motion.

6.3.2.2 Semantic Transform

Semantic transform edits image by an additional computer vision model that provides semantic knowledge while preserving human motion information.

We used Mask-RCNN [133] to implement Semantic Transforms in our experiments. As shown in Fig. 6.3, Mask-RCNN detects and segments regions of people. With this additional model we can obtain a foreground-only video and a background-only video from the original video by superimposing black masks onto background and foreground respectively.

We can test a model on the original video set, foreground-only set and background-only set to get three classifying accuracy Acc_o, Acc_f and Acc_b, then we can compute foreground-only accuracy changing rate and background-only changing rate by Eq. 6.5.

$$\begin{cases} CR_f = \frac{Acc_o - Acc_f}{Acc_o} \\ \\ CR_b = \frac{Acc_o - Acc_b}{Acc_o} \end{cases} \tag{6.5}$$

CR_f is expected to be very close to 0, if the model has really learnt human motion and use human motion to classify. An ideal segmentation of the human-centric foreground well preserves all kinds of information on the person in foreground-only videos, while leaving a silhouette of the person in background-only videos. Therefore, if the model is capable of infering human motion but has no reliance on background, CR_b should have a positive value closer to 1 rather than 0.

6.3.3 3D Transform

A 3D transform is an IPT that directly manipulates nuisance factors during data generation process, as in Eq. 6.6. It has access to every factor in the data and can manipulate each factor separately.

$$T_\theta(I) = G(p, \tau_\theta(\theta)) \tag{6.6}$$

Controlling a certain nuisance factor to change while keeping other factors consistent, we can study how sensitve a model is to this controlled factor, and whether the model's response follow some kind of rule. For example, by changing the viewpoint gradually in different videos and record the classification scores, we can derive the curve reflecting how the model reacts to viewpoint change. If we repeat the viewpoint control experiment on different human appearances, we can further know how the model is influenced by human appearance.

Conclusions drawn from synthetic data can be verified by real data. Synthetic domain and real domain share lots of human motion information that is essential for recognizing human activity. When it is expensive and difficult to collect a large number of control variable videos in real domain , we can instead enlarge the increasing or decreasing step of the controlled variable in different videos, and emphasize on important values that reflect the main trend.

However, the conditions for achieving 3D-transform in reality are very strict. Previously, researchers use robotic arm in a lab setting to control viewpoint and lighting. Such realization of 3D transform is usually expensive and difficult to set up. Recently, due to the popularity of synthetic data, meaning data generated through computer graphics, researchers started to use synthetic data to

control factors of an image, such as viewpoint [131] [132], and material property [22].

We collected a 3D animation dataset from Epic Game marketplace as well as the CMU Mocap dataset [134], and created a toolbox for generating synthetic human activity videos. The synthetic data generation pipeline is built on top of Unreal Engine, a popular game engine for creating 3D video game. The toolbox developed from unrealcv [19] can be used to record and render images. After setting human motion with an 3D animation, we could use the toolbox to configure and control each nuisance factor separately, including the environment, lighting, human appearance and viewpoint.

6.4 Experiment

In this section Identity Preserved Transform is applied to the trained Temporal Segment Networks (TSN) introduced in [123] and trained Inflated 3D ConvNet (I3D) introduced in [124]. TSN is the representative of state-of-the-art video classification models that uses 2D convolutional kernels in neural network. I3D is the pioneer in 3D CNNs, giving rise to many adaptations such as S3D [135], I3D-GCN [136], etc.

We adopted TSN parameters trained on UCF101 [137] provided by OpenMMLab and I3D parameters trained on Kinetics [124] provided by deepmind

publicly on Github. Both models' parameters are pre-trained on ImageNet, input modality to both models is RGB without optical flow.

6.4.1 Implementing Image-space Transfrom

Image-space transform is an Identity Preserve Transform operating on image to simulate nuisance factor change. It can be achieved in the form of image processing and semantic transform.

6.4.1.1 Image Processing Transform

Image processing transform includes image processing techniques widely used for data augmentation. We take 5 common image processing techniques in our experiment. They are applied on UCF101 test set to obtain the top-1 accuracy and top-5 accuracy of TSN classification result on each transformed test set.

Results in Tab. 6.1 shows that the model is susceptible to image processing techniques, even if they preserve complete human motion information. Histogram equalization drops the top-1 accuracy by over 10%, and adding Gaussian noise drops the top-1 accuracy by about 25%. Despite the fact that images transformed by these two techniques still look similar to the original image for human.

The accuracy drop caused by image processing transform conflicts with the

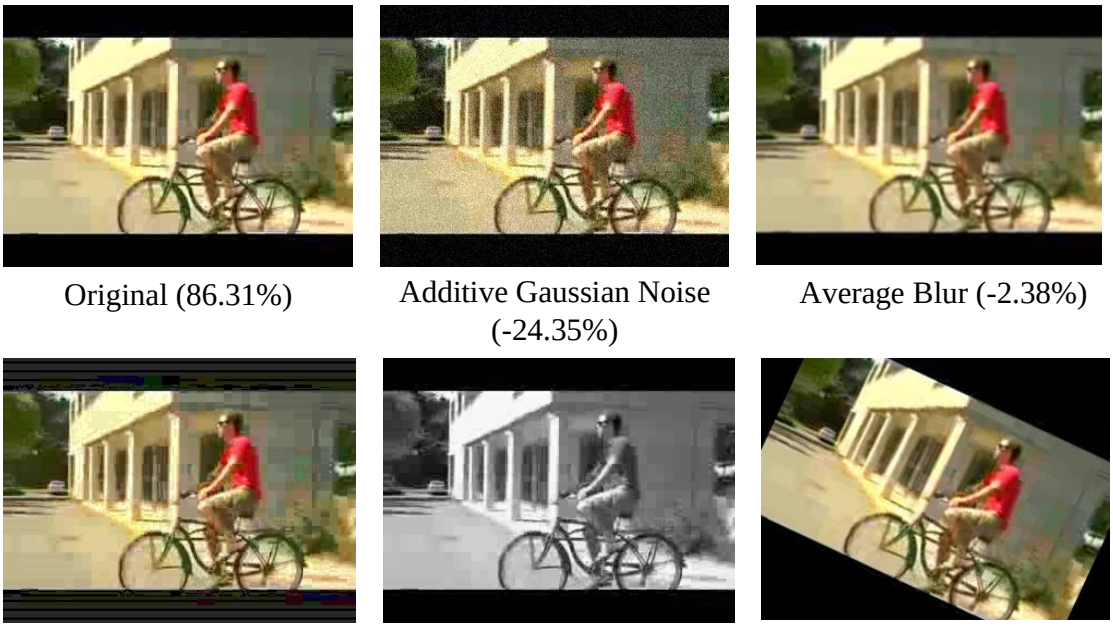

Figure 6.2: Applying Image-space Transform to videos in UCF101. Though these image processing techniques preserve human motion information in the video, they lead to serious dropping of TSN's classification accuracy.

Image-space Transform	Top-1	Top-5
Identical Transform	86.31%	97.99%
Average Blurring	83.93%	96.78%
Histogram Equalization	75.44%	93.39%
Grayscale	71.32%	90.72%
Additive Gaussian Noise	61.96%	85.41%
Clockwise Rotation by 25°	59.11%	80.52%

Table 6.1: Top-1 and top-5 classifying accuracy of trained TSN on a UCF101 test set transformed by 5 different image processing techniques. Identical transform means the video remains as its original version.

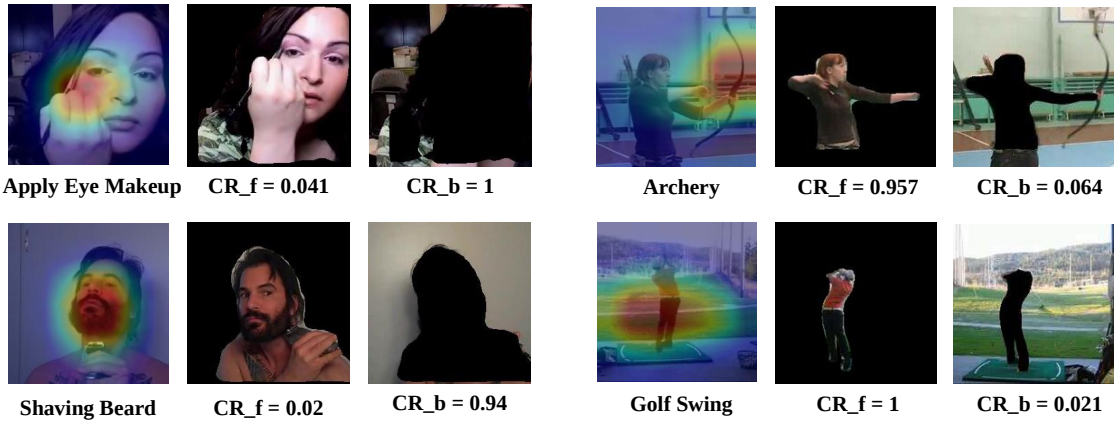

Figure 6.3: Combined analysis with semantic transform and class activation mapping. Left is the case that CR_f being close to 0 while CR_b being close to 1, the model classify these classes by detecting objects, textures, etc. using human motion information in the foreground. Right is the case that CR_b being close to 0 while CR_f being close to 1, the model overfits to background correlated with the activity classes, these classes can be found easily with semantic transform.

expectation for a robust video activity classification model. The reason might be that TSN trained on UCF101 exhibits high reliance on the image pattern exclusive to UCF101.

6.4.1.2 Semantic Transform

Semantic transform edits generated images and videos with a an additional computer vision model that provides prior semantic knowledge. In our experiment we used Mask-RCNN to generate foreground-only videos and background-only videos for UCF101, then tested TSN to get classification accuracy changing rates for each activity class. The segmentation outcomes of Mask-RCNN are reasonably good. For rare cases when Mask-RCNN fails to detect human-

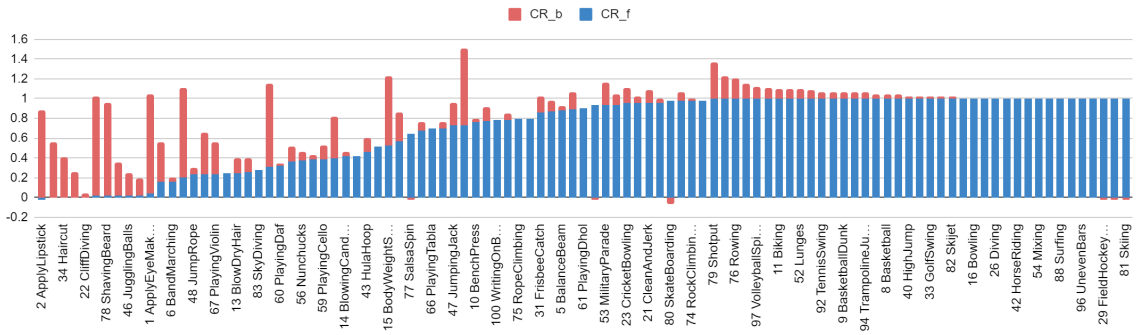

Figure 6.4: Plot of CR_f and CR_b for all classes in UCF101, sorted according to the value of CR_f. The figure clearly shows that most the model has significant performance drop over most activity classes when only given foreground (high CR_f). Only a few activity classes have low CR_f, which we further inspected with CAM [3] visualization.

centric foreground, leading to a full black foreground image and an unmasked background image, we dropped these image pairs. If full black foreground repeatedly appears in one video, we delete video pairs from the test split to ensure the accuracy of our quantitative results.

For a model capable of doing activity classification according to human motion, CR_f is expected to approximate zero while CR_b is expected to have a value closer to 1. However, the real performance is far from perfect. We plotted the CR_f and CR_b given by TSN over all classes of UCF101 and sorted according to the value of CR_f in Fig. 6.4. From Fig. 6.4 it is easy to tell that the model has significant performance drop over most activity classes when only given foreground (high CR_f). Furthermore, there emerges 3 types of results from all UCF101 classes: (1) $1 \approx CR_f \gg CR_b \approx 0$ (2) $0 \approx CR_f \ll CR_b \approx 1$ (3) otherwise.

Emergence of the first type indicates that for these activity classes, the

model has overfit to background rather than learnt human motion. Approximation of CR_b to 0 means the model can classify these activities correctly using only background. Because silhouette of the main person remains in background, if the model can recognize human activities according to the silhouette, it should also achieve high accuracy on foreground-only videos, which better preserves human shape and pose. However, the corresponding CR_f is close to 1, which means accuracy sharply drops on foreground-only videos. Therefore, the model actually does classification using objects and textures in the background.

The quantitative results obtained with semantic transform are supported by class activation mapping [3]. We generated the class activation maps (CAM) corresponding to the class that gets highest classification score for each video. CAMs of activity classes belonging to type (1) show that the model focuses on objects or textures in the background instead of the human silhouette. For example, for Golf Swing the model focuses on the white line on the lawn, and for Archery the model focuses on the bow, as shown in Fig. 6.3.

Simultaneously, the second type of results can be further explained by CAMs. CR_f being close to 0 while CR_b being close to 1 means the model can correctly classify those activity classes with merely foreground, but what specific information is crucial remains unclear. CAMs indicate that the model uses human appearance, objects and textures rather than human motion in the fore-

ground to achieve high accuracy over these classes. For instance, for Apply Eye Makeup, the model focuses the person's face and the eye brush in the foreground, for Shaving Beard it focuses on the person's mouth, as shown in Fig. 6.3.

The analysis above points out that TSN trained on UCF101 classify video activities by detecting objects or textures correlated with each activity class in training data. It has not developed a mechanism to model and infer human motion.

Semantic transform is complementary to class activation mapping. Our approach can be easily scaled up while CAM requires researchers to check a large number of video data with their eyes. Given a large scale video dataset, we can first employ semantic transform to check if the model under examination has overfit to background over some activity classes. After filtering with semantic transform, reviewing work with CAM will significantly decrease.

6.4.2 3D Transform

3D transform is IPT that directly manipulates nuisance factors during data generation process. An ideal 3D transform has access to all factors and can control each one separately, however, it is difficult and expensive to achieve 3D transform on real data. Thus, we first conduct experiments on synthetic data, then verify conclusions drawn from synthetic data with real data.

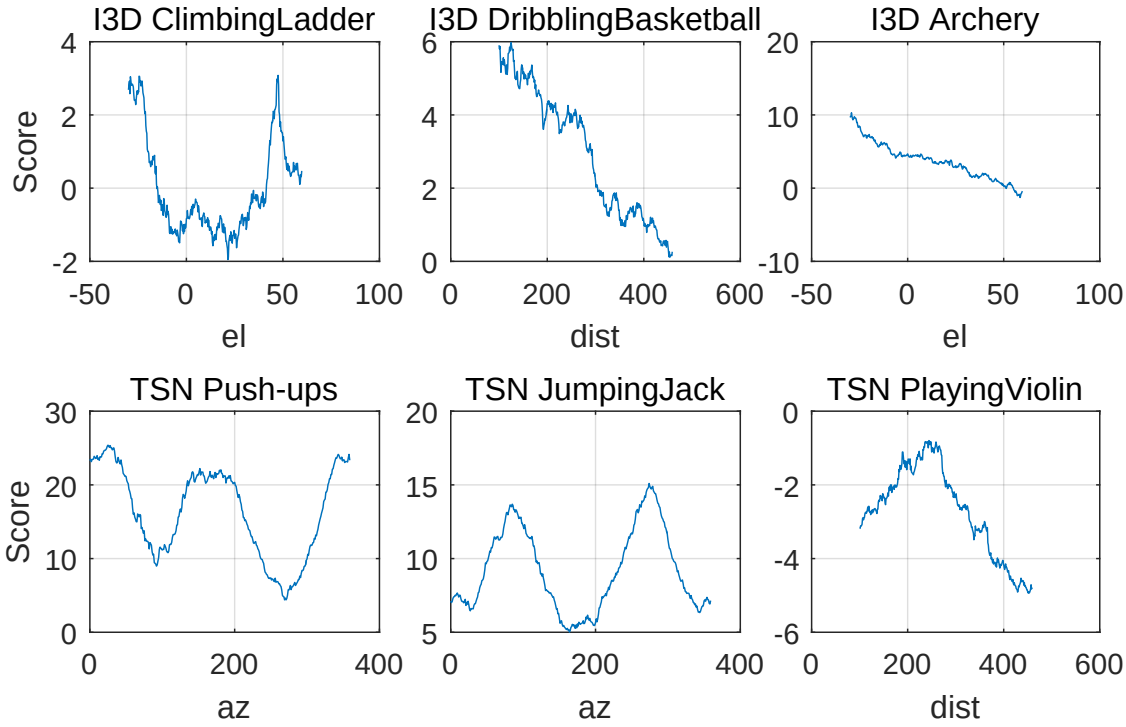

Figure 6.5: Classification score curves corresponding to different controlled factors obtained by 3D transform. If the model is insensitive to viewpoint, the score curve should be stable about a certain score or fluctuate within a relatively small range. However, many score curves yielded by TSN and I3D display trends other than this.

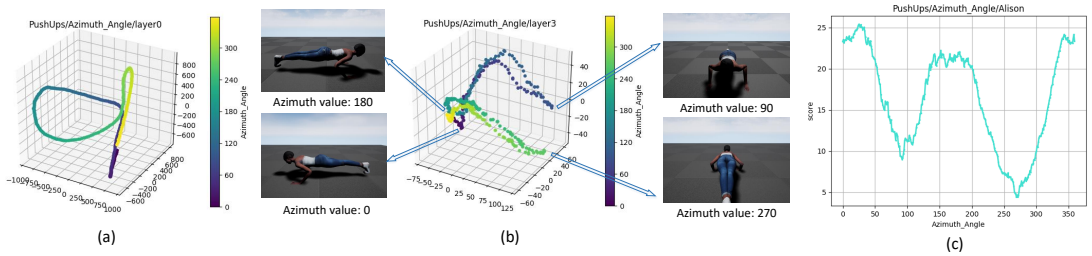

Figure 6.6: TSN's response to the azimuth split over Push-ups. TSN is most likely to recognize Push-ups when viewpoint is set towards the flank of person with invariance to left-right flipping, and it can easily fail to recognize Push-ups when viewpoint is set right towards the person's face or feet.(a) PCA embeddings derived from features output by max-pooling layer after the third Inception Module in the Spatial ConvNet of TSN.(b) PCA embedding derived from features output by Segmental Consensus layer. (c) Curve of classification scores output by TSN over the whole azimuth split.

6.4.2.1 Influence of Viewpoint

In our synthetic video, the camera is set towards the person, thus viewpoint is determined by three variables: camera azimuth angle, camera elevation angle (altitude angle) and camera distance. Keeping two variables constant while increasing the value of the third one iteratively, we generated a synthetic data split for each viewpoint. Simultaneously, other factors, including human motion, background environment, and human appearance are consistent through out a data split.

Record a model's classification score corresponding to each viewpoint variable value over a data split, we can obtain a score curve for this variable, as shown in Fig. 6.5.

If the model is insensitive to viewpoint, the score curve should be stable about a certain score or fluctuate within a relatively small range. However, many score curves yielded by TSN and I3D for different activities are not like this. Instead, there emerged many other trends in these score curves. For example, testing TSN on azimuth split of Jumping Jack videos yields a score curve with two peaks; testing I3D on elevation split of Archery videos yields a score curve monotonically decreasing. Control variable experiment for multiple activity classes indicate that I3D and TSN are sensitive to viewpoint, the sensitivity extent and trend depends on the controlled viewpoint factor and activity class.

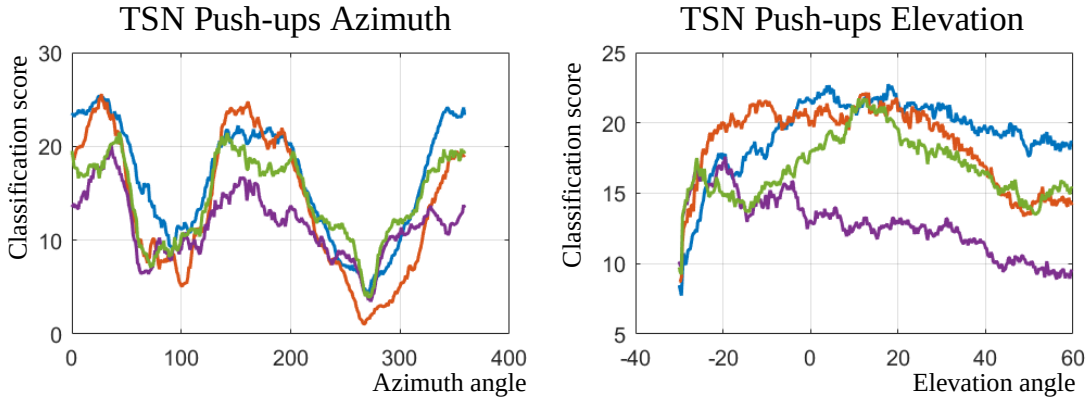

Figure 6.7: Influence of human appearance on the controlled factor score curve. Each color denotes a synthetic human appearance. (a) Though human appearance differs, the TSN classification scores are always high when azimuth is around 0 and 180, while being low when azimuth is at 90 or 270. (b) The score curves obtained from TSN by controlling elevation angle in Push-ups videos look dissimilar on different human appearances.

6.4.2.2 Influence of Human Appearance

The model's response to controlled viewpoint variable can also vary when human appearance changes. We generated the controlled viewpoint variable data splits on different human appearances, then compared the score curves given by the same model over the same activity class.

Appearance	Girl_e	Girl_f	Girl_g	Girl_i
TSN	0.19%	0.46%	7.31%	15.09%
I3D	48.61%	72.50%	70.83%	61.39%
Appearance	Alison	Carla	Claudia	Eric
TSN	48.06%	39.91%	16.20%	11.57%
I3D	51.11%	61.20%	23.24%	48.89%

Table 6.2: Top-1 classification accuracy of TSN and I3D on different synthetic human appearances. We aggregated the azimuth, elevation and distance splits generated on the same human appearance into a larger set, yielding 8 human appearance specific sets in total, then tested with TSN and I3D respectively.

Outcomes show that, in some cases the score curve trend on one human appearance no longer appears on another, for example, the score curves obtained from TSN by controlling elevation angle in Push-ups videos look dissimilar on different human appearances, as shown in Fig. 6.7(b).

However, there also exists cases when the score curves look similar. Take azimuth angle controlled in Push-up videos as example, though human appearance differs, the TSN classification scores are always high when azimuth is around 0 and 180, while being low when azimuth is at 90 or 270, as shown in Fig. 6.7(a).

Classifying accuracy for different human appearances also differs. We computed the top-1 and top-5 classifying accuracy of TSN and I3D on push-ups videos corresponding to eight different human appearances, as reported in Tab. 6.2. Both models show preference for some specific human appearances.

6.4.2.3 Manifold Visualization Using PCA

Similar viewpoint score curves corresponding to many synthetic human appearances could probably reflected the model's classification pattern. Visualizing intermediate layers' features of the model during a forward pass can help us further understand the outcomes of score curves.

In order to better visualize the manifold which impacts the model performance, we extracted features from two layers inside the Spatial ConvNet of

TSN, then applied dimension reduction using Principal Component Analysis to get low-dimensional features for visualization. The lower layer is max-pooling layer after the third Inception Module [123], the higher layer is segmental consensus layer. As Fig. 6.6 shows, 3-dimensional PCA embeddings corresponding to the azimuth angle from 0 to 360 forms a converging manifold, the manifold is twisted through layers between two extraction locations. At Fig. 6.6(b), embeddings of 0 and 180 azimuth angle have been pulled very close, they represent viewpoint set towards the flank of person, while embeddings of 90 and 270 azimuths have transferred to the other side at 3D feature space, they represent viewpoint set straightly towards the person's face or feet.

The visualization indicates high level feature has a certain degree of invariance to viewpoint change. The feature visualization and score curve together proves that TSN has developed a classification pattern for viewpoint in Push-ups videos. After being trained on UCF101 it has correlated push-ups motion with the human pose looking from a viewpoint at the flank, with invariance to left-right flipping.

6.4.2.4 Classification Pattern Supported by Real Data

The classification pattern derived from synthetic data also take effects when the model does classification on real data. Since it is difficult to collect a large number of push-ups videos with different azimuth angles while keeping other

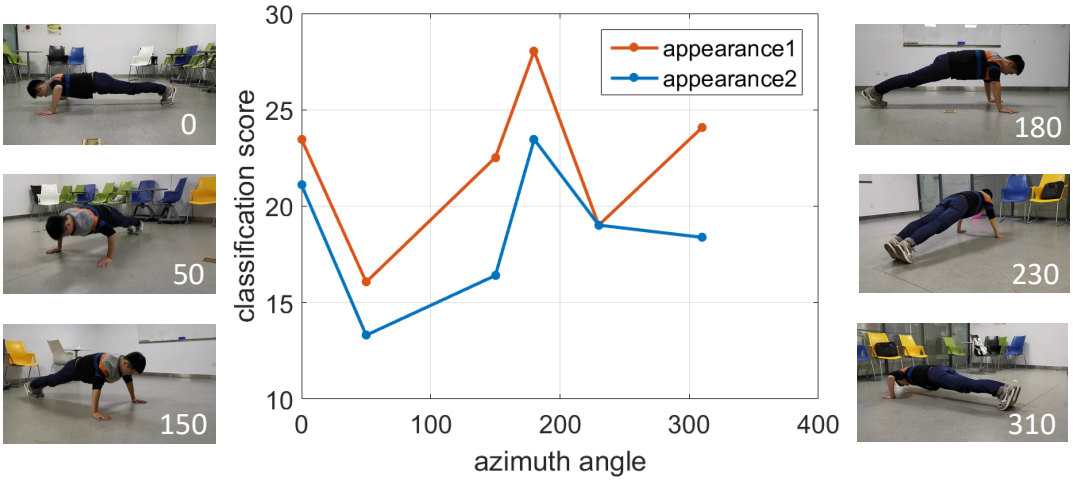

Figure 6.8: Classification score curves for real human Push-ups videos follow the same trend emerged from synthetic human Push-ups videos.

factors constant in the real domain, we first recorded push-ups videos from 6 distinct viewpoint at the same height from the same distance, then found out the azimuth angle each real video corresponds to, they are 0, 50, 150, 180, 230, 310 respectively. As expected, the score curve yielded by TSN for these real videos exhibit a trend similar to those over synthetic data. We conducted azimuth controlled experiment on two real human appearances, both came out to be verified by the classification pattern found with synthetic data.

6.5 Conclusion

With identity preserve transform we proved that TSN, a state-of-the-art video activity classification model pre-trained on ImageNet and trained on

CHAPTER 6. UNDERSTAND WHAT ACTIVITY CLASSIFICATION MODELS HAVE LEARNT

UCF101 is sensitive to factors including the background, related objects, viewpoint and human appearances. Its classification accuracy can be influenced by changes in each of these video factors, therefore TSN probably has memorized the video factor sets for every activity class in UCF101, instead of developing a human motion reasoning mechanism, which is independent of the nuisance factors. Note that our approaches are not only suitable for TSN or I3D. Identity preserve transform is independent of model architecture and thus can be implemented to inspect any trained activity classification model's capabilities, as well as to examine the design of video datasets.

Chapter 7

Conclusion

7.1 Summary

This dissertation summarizes selected research projects in my Ph.D. study. These projects align with a central goal on how to use synthetic data to solve computer vision challenges.

It consists of three parts: 1) data generation infrastructure, 2) model training, 3) model diagnosis.

Chapter 3 and Chapter 4 present two domain adaptation algorithms. One is designed for the case when the accurate geometry of the real object is known in the CAD model. The other is designed for cases where the CAD model appearance is limited compared with real-world in terms of diversity and details. Synthetic data plays a central role in these two projects. Generated ground

truth (dense keypoint annotation, 3D configuration, part segmentation) save enormous human annotation labors.

Chapter 5 and Chapter 6 demonstrate how to utilize the controllability of synthetic data to study the weakness of a model and check whether a model overfits to nuisance factors that are not relevant to the task.

Chapter 1 discusses the difference between real and synthetic data and what tasks can benefit from synthetic data. In this chapter, I will further elaborate future research directions that might benefit from synthetic data.

7.2 Future Directions

The future direction includes three parts. The first is how to build better simulation infrastructure. The second is extensions based on our conducted research. The third are computer vision challenges that can benefit from synthetic data.

7.2.1 Better Simulation Infrastructure

Due to the recent popularity of synthetic data, we can observe the industry devoted resources to building better simulators. What are still missing? CG movies can fool us most of the time with realistic rendering. This proves given enough resources, creating photo-realistic scenes is possible. The difficulty for

computer vision researchers is how to get diverse data in a cost-efficient way. Making the generated data diverse is arguably more important than realism. This diversity is closely related to the domain adaptation and generalization issue.

To get realistic and diverse data, the simulation pipeline requires both diverse 3D models and realistic object behaviors.

Better 3D shape datasets are required. Shapenet [13] is arguably the most important 3D shape dataset for computer vision researchers. But it also has obvious drawbacks for a historical reason. Shapenet is limited by the file format used for storing CAD models. The `obj` format was the standard when Shapenet was created, but it is outdated. It has critical limitations for not capable of representing realistic materials and deformable objects, like a human. New 3D file exchange formats designed for virtual reality applications solved these limitations. Examples are `glTF` from The Khronos Group and `USD` (Universal Scene Description) from Pixar.

Besides better 3D exchange format, the advancement of photogrammetry also makes diverse 3D models more accessible. Shapenet authors built the dataset through crawling man-made CAD models from the 3D warehouse website. But now, photogrammetry enables scanning realistic objects for a very low cost. One noticeable example is MegaScans, which provides free photogrammetry scanned objects. LiDAR camera in iPad pro might also help democratize

photogrammetry, potentially leading to lots of free realistic models. A new 3D dataset with this new level of realism and stores in the new format can significantly benefit the research community.

Besides the model diversity, another diversity is behavior diversity. No matter for autonomous driving or action classification, ensuring the simulator can generate realistic real-world behaviors (*behavior realism*) is important. Different from manually designed car trajectories or human animations, the behaviors copied from the real world is more diverse and have a long-tail distribution. The advancements of computer vision enable the automation of this labor-consuming procedure. Low-cost motion capture systems based on 3D human pose estimation and car pose estimation system from a car-mounted camera are examples for this direction.

7.2.2 Extension Based on Our Work

Topics discussed in this dissertation can be further explored.

The first is domain adaptation. In order to push this direction forward, we need to build more cost-efficient pipeline to generate more diverse data. The domain adaptation problem between real and synthetic data can also be considered as a domain generalization challenge, which requires us to design models that can learn from essential factors related to the task, while avoid overfitting to nuisance factors.

The second topic is combining vision and robotics. Industry robots are precise enough to handle repetitive tasks for many years. But they are limited by the vision module and can not dynamically interact with the world. Vision is one of the bottlenecks limiting robots to automate most repetitive work. Synthetic data can help build a better 3D vision model to help a robot system (robot arm, car) to parse surrounding environments. Simulators have been used for many years in the robotic research, but how to model the complexity of the real world is still a challenge.

The third topic is active data sampling for improving model weakness. In our previous research [22] [131] [138], we demonstrated we can use synthetic data to discover model weakness. We can improve the model through sampling more training data for these weakness. Our preliminary research [139] demonstrated it is effective for 6D object pose estimation, but there are much more can be done.

7.2.3 Challenges Which can Benefit from Synthetic Data

There are many core computer vision challenges can benefit from using synthetic data. They are 1) learn compositional or disentangled representation, 2) learn novel 3D representation, 3) build a knowledge system for vision, 4) learn

through interaction.

First, synthetic data can help develop compositional or disentangled models. Besides driving the performance higher, researchers also want to design models which are explainable, robust, and adaptive. Compositionality is a nice property to have. The compositionality can be in a spatial form, which means combining parts into an object. It can also be at an abstract level (attributes), which means concepts can be combined to create new concepts. This compositionality provides explainable and also makes few-shot learning easier by reusing concepts. Synthetic data can provide this part-whole relationship easily. Attributes of synthetic data can also be tweaked to train and diagnose whether the model has a disentangled representation.

Second, synthetic data can help develop novel 3D shape representation [140] [141]. An internal 3D representation (world model) can have significant benefits for many tasks; it can be used to connect observations from different viewpoints or constrain the smoothness between consecutive frames. Existing shape models, like vertex-based, voxel-based, are successful for CG applications, but they have great limitations for the CV purpose, for example, analysis by synthesis. Synthetic data can provide accurate shapes for diverse scenarios. It can even generate complex and dynamic shapes through physics simulation. This can provide abundant resources that are hard to get from the real world.

Third, synthetic data can help build a knowledge system for vision. Some

knowledge are difficult to learn by observing images and videos. AI can learn naive physics by observing block towers to fail in a virtual world [11]. Synthetic data can also be used for learning affordance tasks, like whether an object can be grabbed, or whether a surface can be sit.

Last but not the least, synthetic data enables learning through interaction with the environment [12]. This is expensive to performe with real simulation. Robotics researchers already prove a visual servo policy and a vision module can be learnt through interacting with the world without explicitly labelled images. The interesting fact is human and all animals learn visual concepts through interacting with the environment rather than given per-pixel annotated images. Given this fact, there are many questions remain to be answered in this direction.

Bibliography

[1] S. Qiao, W. Shen, W. Qiu, C. Liu, and A. Yuille, "Scalenet: Guiding object proposal generation in supermarkets and beyond," in *ICCV*, 2017.

[2] A. Chakrabarti, Y. Xiong, S. J. Gortler, and T. Zickler, "Low-level vision by consensus in a spatial hierarchy of regions," in *CVPR*, 2015.

[3] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *CVPR*, 2016.

[4] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model," *ACM transactions on graphics (TOG)*, vol. 34, no. 6, pp. 1–16, 2015.

[5] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, "Learning from synthetic humans," in *CVPR*, 2017.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[7] D. Xie, T. Shu, S. Todorovic, and S.-C. Zhu, "Learning and inferring "dark matter" and predicting human intents and trajectories in videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[8] P. Wang and A. Yuille, "Doc: Deep occlusion estimation from a single image," in *ECCV*, 2016.

[9] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *CVPR*, 2017.

[10] D. Geman, S. Geman, N. Hallonquist, and L. Younes, "Visual turing test for computer vision systems," *Proceedings of the National Academy of Sciences*, vol. 112, no. 12, pp. 3618–3623, 2015.

[11] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences*, vol. 110, no. 45, pp. 18 327–18 332, 2013.

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17,

no. 1, pp. 1334–1373, 2016.

[13] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[14] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, "How to grow a mind: Statistics, structure, and abstraction," *Science*, vol. 331, no. 6022, pp. 1279–1285, 2011.

[15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *ICCV*, 2015.

[16] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *ECCV*, 2016.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[18] A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto, "Robot learning in homes: Improving generalization and reducing dataset bias," in *NeurIPS*, 2018.

[19] W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim, Y. Wang, and A. Yuille, "Unrealcv: Virtual worlds for computer vision," in *Proceedings of the 25th ACM International Conference on Multimedia*, 2017.

[20] Y. Zuo, W. Qiu, L. Xie, F. Zhong, Y. Wang, and A. L. Yuille, "Craves: Controlling robotic arm with a vision-based economic system," in *CVPR*, 2019.

[21] J. Mu, W. Qiu, G. Hager, and A. Yuille, "Learning from synthetic animals," *CVPR*, 2020.

[22] Y. Zhang, W. Qiu, Q. Chen, X. Hu, and A. Yuille, "Unrealstereo: Controlling hazardous factors to analyze stereo vision," in *3DV*, 2018.

[23] J. Lyu, W. Qiu, and A. Yuille, "Identity preserve transform: Understand what activity classification models have learnt," *CVPR Workshop*, 2020.

[24] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *ECCV*, 2012.

[25] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *CVPR*, 2016.

[26] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," *arXiv preprint arXiv:1603.01312*, 2016.

[27] W. Qiu and A. Yuille, "Unrealcv: Connecting computer vision to unreal engine," in *ECCV Workshop*, 2016.

[28] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, 2018.

[29] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An

open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[30] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[31] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[32] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *ICRA*, 2016.

[33] A. Hundt, V. Jain, C. Paxton, and G. D. Hager, "Training frankenstein's creature to stack: Hypertree architecture search," *arXiv preprint arXiv:1810.11714*, 2018.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[35] G. Csurka, "Domain adaptation for visual applications: A comprehensive survey," *arXiv preprint arXiv:1702.05374*, 2017.

[36] E. Jang, S. Vijaynarasimhan, P. Pastor, J. Ibarz, and S. Levine, "End-to-end learning of semantic grasping," *arXiv preprint arXiv:1707.01932*, 2017.

[37] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, and Y. Wang, "End-to-end active object tracking via reinforcement learning," in *ICML*, 2018.

[38] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, 2018.

[39] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[40] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator." in *International Conference on Intelligent Robots and Systems*, 2004.

[41] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," *arXiv preprint arXiv:1511.07111*, 2015.

[42] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[43] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *ICRA*, 2018.

[44] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-

aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[45] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration," in *ICRA*, 2018.

[46] M. P. Deisenroth, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," *Robotics: Science and Systems*, 2012.

[47] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *CVPR*, 2016.

[48] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, and Y. Wang, "End-to-end active object tracking and its real-world deployment via reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[49] F. Zhong, P. Sun, W. Luo, T. Yan, and Y. Wang, "AD-VAT: An asymmetric dueling mechanism for learning visual active tracking," in *ICLR*, 2019.

[50] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," in *ICCV*, 2015.

[51] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3d orientation learning for 6d object detection from rgb images," in *ECCV*, 2018.

[52] J. Hoffman, D. Wang, F. Yu, and T. Darrell, "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation," *arXiv preprint arXiv:1612.02649*, 2016.

[53] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *International Conference on Intelligent Robots and Systems*, 2017.

[54] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *CVPR Workshop*, 2018.

[55] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," in *ICML*, 2018.

[56] F. Mahmood, R. Chen, and N. J. Durr, "Unsupervised reverse domain adaptation for synthetic medical images via adversarial training," *IEEE Transactions on Medical Imaging*, 2018.

[57] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *CVPR*, 2017.

[58] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discrimi-

native domain adaptation," in *CVPR*, 2017.

[59] C. Li, M. Z. Zia, Q. Tran, X. Yu, G. D. Hager, and M. Chandraker, "Deep supervision with shape concepts for occlusion-aware 3d object parsing," in *CVPR*, 2017.

[60] Z. Hong, Y. Chen, H. Yang, S. Su, T. Shann, Y. Chang, B. H. Ho, C. Tu, T. Hsiao, H. Hsiao, S. Lai, Y. Chang, and C. Lee, "Virtual-to-real: Learning to control in visual semantic segmentation," in *IJCAI*, 2018.

[61] Y. Gao and A. L. Yuille, "Exploiting symmetry and/or manhattan properties for 3d object structure estimation from single and multiple images." in *CVPR*, 2017.

[62] G. Pavlakos, L. Zhu, X. Zhou, and K. Daniilidis, "Learning to estimate 3d human pose and shape from a single color image," in *CVPR*, 2018.

[63] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it smpl: Automatic estimation of 3d human pose and shape from a single image," in *ECCV*, 2016.

[64] M. Omran, C. Lassner, G. Pons-Moll, P. Gehler, and B. Schiele, "Neural body fitting: Unifying deep learning and model based human pose and shape estimation," in *3DV*, 2018.

[65] S. Zuffi, A. Kanazawa, and M. J. Black, "Lions and tigers and bears: Capturing non-rigid, 3d, articulated shape from images," in *CVPR*, 2018.

[66] X. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison*, 2006.

[67] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[68] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *ECCV*, 2016.

[69] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.

[70] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.

[71] Y. Yang and D. Ramanan, "Articulated human detection with flexible mixtures of parts," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2878–2890, 2013.

[72] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *CVPR*, 2014.

[73] B. Sapp and B. Taskar, "Modec: Multimodal decomposable models for human pose estimation," in *CVPR*, 2013.

[74] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield, "Structured domain randomization: Bridg-

ing the reality gap by context-aware synthetic data," in *ICRA*, 2019.
[75] W. Chen, H. Wang, Y. Li, H. Su, Z. Wang, C. Tu, D. Lischinski, D. Cohen-Or, and B. Chen, "Synthesizing training images for boosting human 3d pose estimation," in *3DV*, 2016.
[76] L. Del Pero, S. Ricco, R. Sukthankar, and V. Ferrari, "Articulated motion discovery using pairs of trajectories," in *CVPR*, 2015.
[77] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-ucsd birds 200," 2010.
[78] J. Cao, H. Tang, H.-S. Fang, X. Shen, C. Lu, and Y.-W. Tai, "Cross-domain adaptation for animal pose estimation," in *ICCV*, 2019.
[79] D. Novotny, D. Larlus, and A. Vedaldi, "I have seen enough: Transferring parts across categories," in *BMVC*, 2016.
[80] S. Li, J. Li, W. Lin, and H. Tang, "Amur tiger re-identification in the wild," *arXiv preprint arXiv:1906.05586*, 2019.
[81] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *CVPR*, 2014.
[82] S. Zuffi, A. Kanazawa, T. Berger-Wolf, and M. Black, "Three-d safari: Learning to estimate zebra pose, shape, and texture from images "in the wild"," in *ICCV*, 2019.
[83] B. Biggs, T. Roddick, A. Fitzgibbon, and R. Cipolla, "Creatures great and smal: Recovering the shape and motion of animals from video," in *ACCV*, 2018.
[84] S. Zuffi, A. Kanazawa, D. W. Jacobs, and M. J. Black, "3d menagerie: Modeling the 3d shape and pose of animals," in *CVPR*, 2017.
[85] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *NeurIPS*, 2017.
[86] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in *ECCV*, 2018.
[87] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.
[88] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *ICML*, 2015.
[89] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *ECCV Workshop*, 2016.
[90] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *ICCV*, 2015.
[91] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *NeurIPS*, 2016.
[92] Z. Murez, S. Kolouri, D. Kriegman, R. Ramamoorthi, and K. Kim, "Image to image translation for domain adaptation," in *CVPR*, 2018.

[93] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *ICML Workshop*, 2013.
[94] Y. Kim, J. Yim, J. Yun, and J. Kim, "Nlnl: Negative learning for noisy labels," in *ICCV*, 2019.
[95] Y. Ding, L. Wang, D. Fan, and B. Gong, "A semi-supervised two-stage approach to learning from noisy labels," in *WACV*, 2018.
[96] Y. Zou, Z. Yu, B. Vijaya Kumar, and J. Wang, "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training," in *ECCV*, 2018.
[97] Y. Zou, Z. Yu, X. Liu, B. Kumar, and J. Wang, "Confidence regularized self-training," in *ICCV*, 2019.
[98] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," in *ICLR*, 2017.
[99] G. French, M. Mackiewicz, and M. Fisher, "Self-ensembling for visual domain adaptation," in *ICLR*, 2018.
[100] Y. Li, L. Yuan, and N. Vasconcelos, "Bidirectional learning for domain adaptation of semantic segmentation," in *CVPR*, 2019.
[101] J. Choi, T. Kim, and C. Kim, "Self-ensembling with gan-based data augmentation for domain adaptation in semantic segmentation," in *ICCV*, 2019.
[102] I. Radosavovic, P. Dollár, R. Girshick, G. Gkioxari, and K. He, "Data distillation: Towards omni-supervised learning," in *CVPR*, 2018.
[103] A. RoyChowdhury, P. Chakrabarty, A. Singh, S. Jin, H. Jiang, L. Cao, and E. Learned-Miller, "Automatic adaptation of object detectors to new domains using self-training," in *CVPR*, 2019.
[104] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009.
[105] S. Guo, W. Huang, H. Zhang, C. Zhuang, D. Dong, M. R. Scott, and D. Huang, "Curriculumnet: Weakly supervised learning from large-scale web images," in *ECCV*, 2018.
[106] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann, "Self-paced curriculum learning," in *AAAI*, 2015.
[107] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.
[108] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, 2011.
[109] O. Zendel, M. Murschitz, M. Humenberger, and W. Herzner, "Cv-hazop: Introducing test data validation for computer vision," in *ICCV*, 2015.
[110] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *CVPR*, 2015.

[111] S. Meister, B. Jähne, and D. Kondermann, "Outdoor stereo camera system for the generation of real-world benchmark data sets," *Optical Engineering*, vol. 51, no. 02, p. 021107, 2012.
[112] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.
[113] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *CVPR*, 2016.
[114] G. R. Taylor, A. J. Chosak, and P. C. Brewer, "Ovvv: Using virtual worlds to design and evaluate surveillance systems," in *CVPR*, 2007.
[115] A. Borji, S. Izadi, and L. Itti, "ilab-20m: A large-scale controlled object dataset to investigate deep learning," in *CVPR*, 2016.
[116] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *ACCV*, 2010.
[117] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time weighted median filtering for stereo matching and beyond," in *ICCV*, 2013.
[118] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *CVPR*, 2005.
[119] K. Yamaguchi, D. McAllester, and R. Urtasun, "Efficient joint segmentation, occlusion labeling, stereo and flow estimation," in *ECCV*, 2014.
[120] J. Zbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network," in *CVPR*, 2015.
[121] R. Nair, A. Fitzgibbon, D. Kondermann, and C. Rother, "Reflection modeling for passive stereo," in *ICCV*, 2015.
[122] F. Guney and A. Geiger, "Displets: Resolving stereo ambiguities using object knowledge," in *CVPR*, 2015.
[123] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *ECCV*, 2016.
[124] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *CVPR*, 2017.
[125] B. Zhou, A. Andonian, A. Oliva, and A. Torralba, "Temporal relational reasoning in videos," in *ECCV*, 2018.
[126] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, Y. Yan, L. Brown, Q. Fan, D. Gutfreund, C. Vondrick *et al.*, "Moments in time dataset: one million videos for event understanding," *TPAMI*, 2019.
[127] Y. LeCun, F. J. Huang, L. Bottou *et al.*, "Learning methods for generic object recognition with invariance to pose and lighting," in *CVPR*, 2004.
[128] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, 2017.
[129] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas,

and R. Sayres, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)," *arXiv preprint arXiv:1711.11279*, 2017.

[130] M. Aubry and B. C. Russell, "Understanding deep features with computer-generated imagery," in *ICCV*, 2015.

[131] X. Zeng, C. Liu, Y.-S. Wang, W. Qiu, L. Xie, Y.-W. Tai, C.-K. Tang, and A. L. Yuille, "Adversarial attacks beyond the image space," in *CVPR*, 2019.

[132] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, "Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects," in *CVPR*, 2019.

[133] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *ICCV*, 2017.

[134] Carnegie Mellon Graphics Lab, "Carnegie Mellon University Motion Capture Database," 2016.

[135] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, "Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification," in *ECCV*, 2018.

[136] X. Wang and A. Gupta, "Videos as space-time region graphs," in *ECCV*, 2018.

[137] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[138] M. Shu, C. Liu, W. Qiu, and A. Yuille, "Identifying model weakness with adversarial examiner," *AAAI*, 2020.

[139] Q. Chen, W. Qiu, Y. Zhang, L. Xie, and A. Yuille, "Sampleahead: Online classifier-sampler communication for learning from synthesized data," *BMVC*, 2018.

[140] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[141] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *ECCV*, 2020.

Weichao Qiu

Updated: October 28, 2020 Github: github.com/qiuwch
Email: qiuwch@gmail.com Website: weichaoqiu.com

Education

2016 – 2020 Ph.D. Student in Department of Computer Science, Johns Hopkins University
(Transfered with Prof. Alan Yuille) Advisor: Alan Yuille

2014 – 2016 Ph.D. Student in Department of Statistics, UCLA Advisor: Alan Yuille

2011 – 2014 Master in Department of Electronics and Information Engineering,
Huazhong University of Sci. & Tech. (HUST) Advisor: Xiang Bai

2007 – 2011 Bachelor in Department of Computer Science, HUST

Publications

• Jiteng Mu*, **Weichao Qiu***, Gregory Hager, Alan Yuille, "Learning from Synthetic Animals", CVPR 2020 (Oral) project page

• Jialing Lyu, **Weichao Qiu**, Alan Yuille, "Identity Preserve Transform: Understand What Activity Classification Models Have Learnt", CVPR Workshop 2020

• Michelle Shu, Chenxi Liu, **Weichao Qiu**, Alan Yuille, "Identifying Model Weakness with Adversarial Examiner", AAAI 2020

• Yutong Bai, Qing Liu, Lingxi Xie, Yan Zheng, **Weichao Qiu**, Alan Yuille, "Semantic Part Detection via Matching: Learning to Generalize to Novel Viewpoints from Limited Training Data", ICCV 2019

• Yiming Zuo*, **Weichao Qiu***, Lingxi Xie, Fangwei Zhong, Yizhou Wang, Alan L. Yuille, "Towards Accurate Task Accomplishment with Low-Cost Robotic Arms", CVPR 2019, project page

• Xiaohui Zeng, Chenxi Liu, Yu-Siang Wang, **Weichao Qiu**, Lingxi Xie, Yu-Wing Tai, Chi Keung Tang, Alan Yuille, "Adversarial Attacks Beyond the Image Space", CVPR 2019 (Oral)

• Qingfu Wan, **Weichao Qiu**, Alan Yuille, "Patch-based 3D Human Pose Refinement", CVPR Workshop 2019

• Tae Soo Kim, Mike Peven, **Weichao Qiu**, Alan Yuille, Gregory Hager, "Synthesizing Attributes with Unreal Engine for Fine-grained Activity Analysis", WACV Workshop 2019

• Yi Zhang*, **Weichao Qiu***, Qi Chen, Xiaolin Hu, Alan Yuille, "UnrealStereo: Controlling Hazardous Factors to Analyze Stereo Vision", 3DV 2018 (Oral)

• Qi Chen, **Weichao Qiu**, Yi Zhang, Lingxi Xie, Alan Yuille, "SampleAhead: Online Classifier-Sampler Communication for Learning from Synthesized Data", BMVC 2018 (Oral)

• Siyuan Qiao, Wei Shen, **Weichao Qiu**, Chenxi Liu, Alan Yuille, "ScaleNet: Guiding Object Proposal Generation in Supermarkets and Beyond", ICCV 2017

• **Weichao Qiu**, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, Yizhou Wang, Alan Yuille, "UnrealCV: Virtual Worlds for Computer Vision", ACM MM 2017, project page

• **Weichao Qiu**, Alan Yuille, "UnrealCV: Connecting Computer Vision to Unreal Engine", ECCV VARVAI Workshop 2016

• **Weichao Qiu**, "Generating Human Images and Ground Truth using Computer Graphics", Master Thesis of UCLA, Statistics, 2016

• Jiayi Ma, **Weichao Qiu**, Ji Zhao, Yong Ma, Alan Yuille, Zhuowen Tu, "Robust L2E Estimation of Transformation for Non-Rigid Registration", IEEE Transactions on Signal Processing 2015

• **Weichao Qiu**, Xinggang Wang, Xiang Bai, Alan Yuille, Zhuowen Tu, "Scale-Space SIFT Flow", WACV 2014

* indicates equal contribution

Work Experience

2017.06 - 2017.11	Research Intern of Oculus Research, Pittsburgh (now Facebook Reality Labs)
2009.07 - 2009.10	Software Engineering Intern of Microsoft Research Asia, Innovation Engineering Group

Research Experience

2017.11 - 2020.10	Core team member of DIVA JHU team and developed synthetic activity data generation system. DIVA is an IARPA funded project which focuses on activity detection.
2016.02 - 2020.10	Use Unreal Engine 4 (UE4) to construct virtual worlds for training and diagnosing computer vision algorithms. Developed an UE4 plugin `UnrealCV` (unrealcv.org) to make realistic virtual worlds easier to create, use and share.
2015.06 - 2016.01	Use Blender to generate synthetic human images with rich annotation, including keypoint and semantic part labeling.
2013.02 - 2013.11	Visiting Graduate Researcher, Statistics Department, UCLA
2010 - 2013	Research Assistant, Multimedia and Communication Lab (MCLab), HUST

Other Experience

• Live project demo in CVPR 2017 (unrealcv project, unrealcv.org)
and CVPR 2019 (low-cost robotic arm project, craves.ai)

• Reviewer of PAMI, TIP, SIGGRAPH, AAAI, IJCAI, CVPR, ICCV, ECCV

• Maintainer of github repo (unrealcv/synthetic-computer-vision), a collection of resource related to synthetic data generation.

• Instructor of intersession course, Virtual Reality App Development (JHU 2018)
A three-week intersession course (jhu-vr.github.io) for undergraduates, which is designed from ground up by me. 45+ undergraduates enrolled and gave good feedback.

• TA of Vision as Bayesian Inference (JHU 2018), Probabilistic Models of the Visual Cortex (UCLA 2015), Software Engineering, Image Processing (HUST 2011, 2012)

• 2nd Place of Design for Development Award, Microsoft Imagine Cup Worldwide Final, Cairo, Egypt (2009)

• President of Unique Studio, HUST (hustunique.com), which is a student organization focused on software development (2009 - 2010)